Graceful Degradation and Speculation for Robots in Highly Dynamic Environments

Marjorie Bournat

Thèse présentée pour obtenir le grade de docteur de
Sorbonne Université

Laboratoire d'Informatique de Paris 6
École Doctorale Informatique, Télécommunications et Électronique (ED130)

Discipline : Informatique

Graceful Degradation and Speculation for Robots in Highly Dynamic Environments

Dégradation progressive et spéculation pour les robots dans des environnements hautement dynamiques

---

Marjorie Bournat

Rapporteurs :

Ralf Klasing, Directeur de recherche CNRS, LaBRI

Roger Wattenhofer, Full Professor, Swiss Federal Institute of Technology in Zurich

Examinateurs :

Lélia Blin, Maîtresse de conférences (HDR), Sorbonne Université

Sylvie Delaët, Maîtresse de conférences (HDR), Université Paris Sud

Carole Delporte, Professeure des universités, Université Paris Diderot

Directeur de thèse :

Franck Petit, Professeur des universités, Sorbonne Université

Encadrants de thèse :

Yoann Dieudonné, Maître de conférences, Université de Picardie Jules Verne

Swan Dubois, Maître de conférences, Sorbonne Université

Remerciements

Pour lire ces remerciements vous pouvez écouter “Nothing like you and I” de The Perishers, ça donnera un petit côté nostalgique à la chose (ou pas, ça dépend de la sensibilité de chacun).

En tout premier, je voudrais remercier Yoann Dieudonné, Swan Dubois, et Franck Petit de m'avoir encadrée durant ma thèse et de m'avoir poussée à produire des meilleures versions de mes travaux pour pouvoir obtenir ce qui, je crois, est une belle thèse.

En particulier, je voudrais remercier Swan Dubois qui, tout au long de cette thèse, m'a soutenue tant au niveau scientifique qu'au niveau humain. Je suis contente de l'avoir eu en tant qu'encadrant de thèse, et d'avoir pu partager sa vision de la vie pendant des heures durant. Parmi ses leçons de vie, je crois qu'il aimerait que je cite “La vie est une succession de choix et il faut en assumer les conséquences”.

Je voudrais aussi remercier les membres du jury Lélia Blin, Sylvie Delaët, et Carole Delporte d'avoir bien voulu examiner ma thèse, et Ralf Klasing et Roger Wattenhofer d'avoir bien voulu lire et rapporter cette thèse.

Je tiens aussi à remercier mes amis du LIP6 : Anaïs, Antoine, Arnaud, Cédric, Damien, Daniel, Darius, Denis, Etienne, Florent, Francis, Gauthier, Guillaume, Hakan, Ilyas, Illyou, Joao, Jonathan L., Jonathan S., Julien, Laure, Laurent, Léonardo, Lucas, Luciana, Ludovic, Lyes, Mathieu, Maxime B., Maxime L., Maxime V., Pierre, Redha, Rudyar, Saalik, Sébastien, Sreeja, Thibault, et Vincent avec qui on a bien rigolé, joué aux cartes, fait des pauses interminables pour décompresser, et passé de super moments durant les conférences.

Je voudrais remercier également ma famille : ma maman, mon papa, ma sœur et mes nièces qui ont toujours été là pour moi dans les moments heureux comme moins heureux, et qui m'ont toujours soutenue quels que soient mes choix.

Enfin, je voudrais remercier mon Bibou d'amour d'être à mes côtés, et de me permettre de vivre chaque jour des matins d'été ensoleillés.

À ma maman, et à mon papa,
À ma sœur Laure,
À mes nièces Joy, Lindsay et Ménehould,
À mon Bibou,

Abstract

Distributed systems are systems composed of multiple communicant processes cooperating to solve a common task. This is a generic model for numerous real systems as wired or mobile networks, shared-memory multiprocessor systems, and so on. From an algorithmic point of view, it is well-known that strong assumptions (as asynchronism or mobility) on such systems lead often to impossibility results or high lower bounds on complexity. In this thesis, we study algorithms that *adapt themselves* to their environment (i.e., the union of all assumptions on the system) by focusing on the two following approaches. *Graceful degradation* circumvents impossibility results by degrading the properties offered by the algorithm as the environment become stronger. *Speculation* allows to bypass high lower bounds on complexity by optimizing the algorithm only on more probable environments.

Robot networks are a particular case of distributed systems where processes are endowed with sensors and able to move from a location to another. We consider *dynamic environments* in which this ability may evolve with time. This thesis answers positively to the open question whether it is possible and attractive to apply gracefully degrading and speculative approaches to robot networks in dynamic environments. This answer is obtained through contributions on gracefully degrading *gathering* (where all robots have to meet on the same location in finite time) and on speculative *perpetual exploration* (where robots must visit infinitely often each location).

Résumé

Les *systèmes distribués* sont des systèmes composés de plusieurs processus communiquants et coopérants ensemble pour résoudre des tâches communes. C'est un modèle générique pour de nombreux systèmes réels comme les réseaux sans fil ou mobiles, les systèmes multiprocesseurs à mémoire partagée, *etc.* D'un point de vue algorithmique, il est reconnu que de fortes hypothèses (comme l'asynchronisme ou la mobilité) sur de tels systèmes mènent souvent à des résultats d'impossibilité ou à de fortes bornes inférieures sur les complexités. Dans cette thèse, nous étudions des algorithmes qui *s'auto-adaptent* à leur environnement (i.e., l'union de toutes les hypothèses sur le système) en se concentrant sur les deux approches suivantes. La *dégradation progressive* contourne les résultats d'impossibilité en dégradant les propriétés offertes par l'algorithme lorsque l'environnement devient fort. La *spéculation* contourne les bornes inférieures élevées sur les complexités en optimisant l'algorithme seulement sur les environnements les plus probables.

Les *réseaux de robots* représentent un cas particulier des systèmes distribués où les processus, dotés de capteurs, sont capables de bouger d'une localisation à une autre. Nous considérons des *environnements dynamiques* dans lesquels cette capacité peut évoluer avec le temps. Cette thèse répond positivement à la question ouverte de savoir s'il est possible et bénéfique d'appliquer les approches progressivement dégradante et spéculative aux réseaux de robots dans des environnements dynamiques. Cette réponse est obtenue en étudiant le *rassemblement* (où tous les robots doivent se retrouver à la même localisation en temps fini) progressivement dégradant et l'exploration perpétuelle (où les robots doivent visiter infiniment souvent chaque localisation) spéculative.

Contents

List of Figures

List of Tables

List of Algorithms

Index

Part I

Context

Chapter 1

Introduction

Contents

In this thesis, we study systems that are composed of multiple processes able to communicate together. Such systems are said to be distributed. Each process of a *distributed system* executes a (local) algorithm (i.e., ordered sequence of instructions) in an uncoordinated manner, i.e., without the help of any central entity. A distributed algorithm is the set of all the local algorithms of the entities of a distributed system.

Distributed systems are opposed to centralized systems in which only one entity has a global view of the system and makes all the decisions. In distributed systems, all the processes make their own decisions based on their own partial knowledge of the system. They have their own clock, and compute at their own speed. In order to solve problems, the processes have to cooperate all together even though they have not the same perception of time, the same vision of the system...

Even if their design may be tricky on various aspects, in particular the difficulty of synchronizing local algorithms, distributed systems present some advantages. Indeed, the processes can proceed in parallel, making tasks executions faster than if they were performed by a single process. Besides, since there are multiple processes to execute a task, it is possible to tolerate faults: if one of the processes stops to execute its algorithm correctly it could still be possible for the problem to be solved.

A distributed system is a general model that permits to represent various real systems like phone networks, transportation networks, internet. There exist multiple assumptions that can be made on a distributed system to represent all those various real systems: Are the entities synchronous (their computational speed is bounded) or are they asynchronous (their computational speed is finite but not bounded)? Are the entities able to communicate? How do they communicate (thanks to shared memory or thanks to messages sent)? Are the entities able to communicate synchronously (the time to route the messages is bounded) or asynchronously (the time to route the messages is finite but unbounded)? Can entities be subject to faults? Is the graph of communication of the entities represented thanks to a static graph or a dynamic graph (where edges model the possibility for two entities to communicate, and may appear and disappear with time)?

An environment corresponds to the set of assumptions that are made on a distributed system. Some environments are harder than others: for instance an environment composed of asynchronous entities is harder than the same environment where the entities are synchronous. When an environment is too harsh, some problems become impossible to solve or lower bounds (i.e., the minimum performance according to some metrics like time used, memory used, number of messages sent, *etc.*, necessary to solve a problem) to solve some problems increase. For instance, the consensus problem (where processes have to decide irrevocably, in finite time, the same value among a set of values initially proposed by each of the processes) is impossible when the processes are asynchronous and may stop the execution of their algorithm [82].

While considering distributed algorithms the classical approach is to analyze the feasibility of problems, i.e., determine the environments in which the problems studied are solvable and the ones in which they are impossible to solve. Once these environments have been determined, the

classical approach consists in finding at least one algorithm per environment where the problems are solvable and this preferably with good performance (like good time complexity). An algorithm conceived in a particular environment does not guarantee any correct behavior or good performances when executed in another environment. This approach may be restrictive, e.g., in a context where we cannot know in advance in which environment the algorithm will be actually executed. This is particularly true when the algorithm is executed in an environment that may change with time, like for instance an environment where processes may be synchronous at some time and asynchronous at some other time or an environment that is more or less dynamic (i.e., the frequency of the appearance and disappearance of the edges of the graph of communication of the processes are more or less high) depending on the time.

In this thesis, we adopt an alternative approach by studying algorithms that are self-adaptive to the environment in which they are executed. We focus on two such approaches: the *indulgence/graceful degradation* and the *speculation* that we describe below.

Indulgent/Gracefully degrading algorithms. Indulgent algorithms [5, 118, 131] and gracefully degrading algorithms [20] (that we define afterwards) share the same underlying idea but applied to different environments: indulgent algorithms focus on environments in which only the synchronicity changes, while gracefully degrading algorithms focus on environments in which only the dynamics of the system change. When such algorithms are executed in an environment in which the problem studied is unsolvable, they obviously do not solve the problem itself but guarantee a best effort strategy. Indeed, they guarantee that as long as the conditions of the system are not suitable to solve the problem then they solve a degraded version of it.

For instance, Biely et al. [20] presented a gracefully degrading algorithm that solves the consensus problem in systems where the connectivity assumptions of the graph of communication are strong (the edges are frequently present), and that gracefully degrades to k-set agreement (where processes have to decide irrevocably, in finite time, k values among a set of values initially proposed by each of the processes) when the connectivity between the communicating processes decreases and makes the consensus problem impossible.

More precisely, indulgent/gracefully degrading algorithms solve the problem $\mathcal{P}$ studied when they are executed in some environments in which this problem is solvable and they solve weaker problems than $\mathcal{P}$ otherwise. In distributed computing, the definitions of problems are generally decomposed into a safety and a liveness property. Intuitively, according to Lamport [117] "a safety property is one which states that something will not happen" and "a liveness property is one which states that something must happen." The problems weaker than $\mathcal{P}$, solved by an indulgent/gracefully degrading algorithm, may ensure the safety of $\mathcal{P}$ to preserve the essence of this problem.

These approaches are a way to circumvent impossibility results that occur in some environments since indulgent/gracefully degrading algorithms provide a best effort solution to the problem when executed in an environment in which the problem is unsolvable. Such algorithms are also useful when the environment of the system in which they will be executed is not known in advance, or when the environment may evolve with time since they will adapt to these changes without any external help by providing the best possible properties in each environment.

Speculative algorithms. In distributed computing, the classical approach to evaluate the efficiency of an algorithm is to consider the worst case in order to provide upper bound (i.e., the maximum performance according to some metrics like time used, memory used, number of messages sent, *etc.*, necessary to solve a problem) for any possible execution in a given environment. The speculative approach [114, 77, 9] is based on the observation that the worst environment possible in which a problem is solvable is not always the most frequent one. When the environments in which the algorithms are more often executed do not correspond to the worst case possible, the classical worst-case approach may provide upper bounds that are very far from the "practical"

ones. To alleviate this problem, the speculative approach consists in optimizing the algorithms in the most frequent environment in which they will be executed. Note that the analysis of the efficiency of a speculative algorithm is still with respect to the worst case in order to provide an upper bound for any execution in the most probable actual environment.

More precisely, speculative algorithms solve the problem studied in any environment in which they are susceptible to be executed but also they are more efficient when executed in the environments that are more probable in practice. The optimization provided by a speculative algorithm is, depending on the context, on time complexity, memory, number of exchanged messages, ...

This approach is a way to circumvent important lower bounds obtained in some rare environments since speculative algorithms provide an optimized solution to the problem when executed in environments in which the problem is frequently executed (i.e., the upper bound obtained in the most frequent environments is lower than the lower bound obtained while considering all the possible executions).

1.1 Robot Networks in Dynamic Environments

In this thesis, we are interested in robot networks [136], i.e., networks made of moving processes endowed with sensors to sense their environment. We consider cohort of robots with few capacities. Numerous potential applications exist for such multi-robot systems: surrounding, patrolling, exploration of various environments, *etc.*, moving and sensory capabilities are key ingredients of such distributed systems to achieve collaborative tasks. In this thesis, we consider distributed systems with assumptions and robot capabilities as weak as possible to determine the feasibility of some problems in given settings.

A first "natural" approach when studying robot networks is to consider robots evolving in the continuous space [45, 72, 139, 123]. A second approach consists of discrete environments modeled by static graphs where nodes represent the locations where the robots may be located and edges represent the possibility for a robot to move from one location to another one [85, 17, 64, 69].

However all the environments are not static: some environments are dynamic and may be represented by dynamic graphs in which nodes and edges may appear and disappear with time. Indeed, dynamic graphs are useful to represent unstable environments that may change over time, like for instance, a transportation network, a building in which doors are closed and open over time, or streets that are closed over time due to work in process or traffic jam in a town.

In this thesis, we consider robots evolving in dynamic graphs composed of a fixed set of nodes that model the locations where robots may be located, and a set of edges that may appear and disappear with time and represent the possibility for a robot to move from one location to another one at a given time.

There exist multiple systems that can be modeled thanks to dynamic graphs, all these systems do not have the same connectivity over time (i.e., frequencies of appearance and disappearance of the edges). For instance, public transportation networks have a periodic connectivity that is induced by the movements of the transporters while phone networks have an uncontrolled connectivity. Hence, to represent the different connectivity assumptions, there exist different classes of dynamic graphs [40, 38].

When speed and scheduling of topological changes vary, some problems become impossible or lower bounds to solve some problems increase. This motivates gracefully degrading and speculative approaches.

Problematic of the thesis. As the gracefully degrading and speculative approaches have never been applied to robot networks, the objective followed in this thesis is to determine whether it is tractable and worthwhile to do so.

Indeed, the strong restrictions of this original model do not speak in favor of a positive answer at first glance. In particular, in a traditional distributed system, processes exchange information

by sending messages (that may be duplicated to overcome temporary or permanent edge absence without any impact on the ongoing behavior of processes) whereas robots have to move to communicate (directly or not) with each other (without the ability to duplicate themselves leading to the necessity to manage cautiously the robots moves according to the dynamics of the environment). Hence, designing an algorithm for a robot network able to adapt to several environments seems to be a challenging task.

The contribution of this thesis is to overcome this challenge and to provide a positive answer to this question by focusing on two classical problems of the literature: the gathering (in which all the robots have to be located in finite and bounded time on the same node of the graph) and the perpetual exploration (in which all the nodes of the graph have to be visited infinitely often by at least one robot) problems.

1.2 Contributions and Outline of the Thesis

In this section, we present for each part presented in this thesis its outline as well as the contributions detailed in it.

Context. We present in Part I the background needed to understand this thesis.

More precisely, in Chapter 2, we describe different representations of dynamic graphs, the generalization of the notion of path in static graphs to dynamic graphs, and then based on this notion, we present the hierarchy of classes of dynamic graphs of Casteigts et al. [40, 38] that permit to compare them. Finally, we present the model of dynamic graphs we use in this thesis.

Chapter 3 is a chapter dedicated to the presentation of robots. In this chapter, we detail the computational model of the robots, the different environments in which they may evolve (continuous space, static graphs, dynamic graphs), the different capacities they may be endowed with, and the classical problems that are solved using robots. Finally, we present the model of robots used in this thesis.

The main definitions explaining algorithms and impossibility results for robots evolving in dynamic graphs are given in Chapter 4. In this chapter, we also present a general framework introduced by Braud-Santoni et al. [34] that helps to prove impossibilities results in dynamic graphs.

Graceful degradation. Part II presents our first contribution of this thesis: we show that it is possible to apply the gracefully degrading approach to robot networks. This is done by studying the gathering problem in dynamic rings. This problem is impossible to solve in highly dynamic rings (dynamic rings in which at most one edge may be missing forever from a given time); therefore it represents a good case study for the graceful degradation.

In Chapter 5, we give a formal definition of the gracefully degrading notion applied to robots evolving in dynamic graphs. Then we describe the related work that adopts an indulgent or a gracefully degrading approach. None of these algorithms consider robots evolving in dynamic graphs. We decide to apply a gracefully degrading approach to robot networks evolving in dynamic graphs in studying the gathering problem since this problem is impossible to solve in highly dynamic graphs. Finally, we present articles dealing with the gathering problem in dynamic graphs.

Then Chapter 6, provides a gracefully degrading gathering algorithm. This algorithm solves the gathering problem in dynamic graphs where this problem is solvable, and degrades it (solves weaker versions of the gathering problem) when it is executed in dynamic graphs in which the gathering problem is not solvable. The degradation we propose is done on the liveness of the problem, not on its safety. The safety of the gathering problem imposes that all robots that terminate do so on the same node, and the liveness imposes that every robot terminates in finite and bounded time. In the degradation we propose the liveness is weakened such that at most one

robot may not terminate or (not exclusively) all robots that terminate do so eventually (and not in a bounded time). Note that it is the same algorithm that is executed in each of the dynamics possible, and that our algorithm has no means to detect the dynamics in which it is executed.

Finally, Chapter 7 concludes the part indicating some interesting perspectives to this study. In particular, we extend the notion of gracefully degrading algorithms: we present optimum gracefully degrading algorithms which are gracefully degrading algorithms that provide the best version of the problem considered in the environments in which they are executed.

Speculation. Part III gives our second contribution of this thesis: it is possible to apply the speculative approach to robot networks. This is shown thanks to the study of the perpetual exploration problem in dynamic graphs.

In Chapter 8, we propose a formal definition of the speculative notion applied to robots evolving in dynamic graphs. Then we describe the related work that adopts a speculative approach. None of these algorithms use robots evolving in dynamic graphs. To extend speculation to robot networks evolving in dynamic graphs, we decide to study the exploration problem. Finally, we present articles dealing with the exploration problem in dynamic graphs.

In Chapter 9, we first characterize the necessary and sufficient number of (non-faulty) robots that permit to solve the perpetual exploration problem in highly dynamic rings. While studying the sufficiency, we provide a speculative algorithm. This algorithm is speculative considering the cover time which is the worst time of the minimal time taken by the robots to explore at least once each node of the ring from any time in all the possible executions of their algorithm. Indeed, it solves the perpetual exploration problem in an unbounded cover time in highly dynamic rings, but in a bounded and asymptotically optimal cover time in static rings.

As indicated, we consider robots with few capacities. However, even when considering robots with very few capacities, it is still possible to have some robots that are subject to faults. In this thesis, we particularly study robots subject to transient faults which are arbitrary faults such that there exists a time from which these faults do not occur anymore. Algorithms that tolerate transient faults are called self-stabilizing algorithms. When a self-stabilizing algorithm uses robots, they are called self-stabilizing robots. Self-stabilizing algorithms are very powerful algorithms since they solve the problem studied even after the occurrence of very important damages in the system. Note that there is no self-stabilizing algorithm solving the exploration problem neither in static nor in dynamic graphs.

In Chapter 10, we perform a similar study as in Chapter 9 but considering self-stabilizing robots. Indeed, we characterize the necessary and sufficient number of self-stabilizing robots that permit to solve the perpetual exploration problem in highly dynamic rings; and while analyzing the sufficiency, we provide a speculative algorithm that solves the perpetual exploration problem with an unbounded cover time in highly dynamic rings and with a bounded and asymptotically optimal cover time in static rings.

Chapter 11 concludes Part III, giving a generalization of our work on the perpetual exploration problem in highly dynamic rings to any highly dynamic graphs, and presenting some interesting future work to do. In particular, we extend the notion of speculative algorithms: we present optimum speculative algorithms which are speculative algorithms whose complexities reach the lower bounds for the problem considered in the environments in which they are executed.

Conclusion of the Thesis. Chapter 12 concludes the thesis giving an overview of the main results. In particular the goal of this thesis was to determine whether gracefully degrading and speculative approaches could be applied to robot networks evolving in dynamic graphs, and we answered positively to this question, defining these notions formally in this precise context, and presenting two study cases: the gathering and the exploration problem in dynamic graphs. Then we detail the perspectives to accomplish which basically consist in finding new complexity measures to apply to robot networks evolving in dynamic graphs to improve the conception of spec-

ulative algorithms using robots evolving in dynamic graphs.

Appendix. In Chapter A we give an overview of another work we did that is related to robot networks but that is not related to the gracefully degrading or the speculative approaches. This work deals with the approach problem which consists for two robots evolving in the plane, endowed with a limited visibility range, to be located, in finite time, in the range of vision of each other. We provide a polynomial solution (in terms of movements) to this problem while considering asynchronous robots endowed with few capacities to orient themselves in the plane.

Finally, in Chapter B, we give a French summary of the thesis.

Publications. Results developed in this thesis have been published in various forums. The gracefully degrading gathering algorithm presented in Chapter 6 appeared in the proceedings of *International Symposium on Stabilization, Safety, and Security of Distributed Systems* 2018 [31]. The speculative perpetual exploration algorithm using non-faulty robots detailed in Chapter 9 has been published in the proceedings of *IEEE International Conference on Distributed Computing Systems* 2017 [29], and in *rencontres francophones sur les Aspects Algorithmiques des Telecommunications* 2017 [30]. The self-stabilizing speculative perpetual exploration algorithm described in Chapter 10 has been published in the proceedings of *International Symposium on Stabilization, Safety, and Security of Distributed Systems* 2016 [27], and in *Theoretical Computer Science* [28].

The work sketched in Chapter A has been published in the proceedings of *International Symposium on Distributed Computing* 2017 [22], in *Distributed Computing* [24] and in *rencontres francophones sur les Aspects Algorithmiques des Telecommunications* 2018 [23].

Chapter 2

Dynamic Graphs

Contents

Few decades ago, most of distributed systems were modeled thanks to static graphs where the nodes represent the processes and the edges represent the possibility for a process to communicate with another one. Similarly, when considering robot networks, generally robots evolve in static graphs where the nodes represent the locations where robots may be located and the edges represent the possibility for a robot to move from one location to another one.

However, nowadays, there are more and more mobile processes. Hence, the communications are not all static. Indeed, for instance, when two processes roll away, their wifi communications may be lost. Similarly, nodes are not always connected, some nodes may be disconnected at some times. Likewise, the environments in which robots evolve are not all static. Indeed, it is possible for robots to move in environments that vary with time. For instance, robots may evolve in an environment modeled by a graph, where the edges represent roads that are closed and opened depending on the traffic jam or work in process. We can also model robots moving in a subway, the edges of the graph in this case represent the metro going from one station to another one. It is also possible for robots to evolve in an environment in which a disaster has occurred, and where the locations where the robots may be located and the roads they may borrow are unstable and may be sometimes inaccessible. Hence, robots may evolve in many environments changing with time. In conclusion, static graphs are not sufficient to model all the possible systems.

To model all these dynamic systems, multiple models of dynamic graphs have been introduced depending on what evolves with time (nodes, edges, both, *etc.*). The challenge is to formally generalize well-known definitions of static graphs that are no longer well defined for dynamic graphs (like for instance a path). In this chapter, we first present, in Section 2.1, different models and dynamics of graphs. Then, in Section 2.2, we describe our model.

2.1 State of the Art

Dynamic graphs have been studied by multiple authors. Multiple models have been defined to represent them. We give, in Section 2.1.1, some of these representations. Then, in Section 2.1.2 we present a hierarchy that classifies the dynamic graphs depending on their dynamics.

2.1.1 Repesentations of Dynamic Graphs

In this section, we present some representations used to model dynamic graphs. We only present the models of dynamic graphs that are general and may be used in various contexts. For instance, among the models we do not present here, there is the PV-graph model [89] which can represent only dynamic graphs where each edge appears periodically (refer to Section 8.2.1 for more details about this model).

Temporal network. Kempe et al. [112] introduce the term temporal networks to define dynamic graphs. A temporal graph is a dynamic graph (G, λ) where λ is a function that associates to each edge multiple reals corresponding to the time at which the two extremities of this edge may interact.

This implies that these kinds of graphs permit to model dynamic graphs in which edges may appear and disappear with time.

Evolving graph. Ferreira [80] introduces the notion of evolving graphs which permits to represent dynamic graphs thanks to a sequence of static graphs. More precisely, an evolving graph $\mathcal{G}$ is represented thanks to a couple (G, S_G), where $G = (V, E)$ is a static graph with V the set of its nodes and E the set of its edges, and S_G is a sequence of subgraphs of G. At each instant time i, the dynamic graph $\mathcal{G}$ is described by the static graph $G_i = (V_i, E_i) \in S_G$.

Note that this representation of dynamic graphs is richer than the one of temporal networks since it permits to model dynamic graphs where nodes and edges may appear and disappear with time.

Evolving graphs are a very intuitive and easy manner to represent dynamic graphs. In this thesis, we choose to consider evolving graphs to represent dynamic graphs.

Stream graph. Recently, Latapy et al. [120] present another model to represent dynamic graphs called stream graphs. A stream graph $\mathcal{G}$ is defined by a quadruplet (T, V, W, E) where:

- T is either a discrete interval of time included in $\mathbb{N}$ (for a discrete time system), or a continuous interval of time included in $\mathbb{R}^+$ (for a continuous time system).
- V is a set of nodes.
- W is a set of temporal nodes. A temporal node $w = (\tau, u) \in W$ (with $\tau \subseteq T$ and $u \in V$) indicates that the node u is present in the graph at each instant time $t \in \tau$.
- E is a set of temporal links. A temporal link $e = (\tau, (u, v)) \in E$ (with $\tau \subseteq T$, $u \in V$, $v \in V$, and $u \neq v$) represents an edge between two distinct nodes u and v present in the graph at each instant time $t \in \tau$. Note that necessarily $(\tau, u) \in W$ and $(\tau, v) \in W$ (i.e., the two extremities of the edge must be present in the graph at least at each instant time $t \in \tau$); otherwise the edge cannot exist.

Latapy et al. have redefined in their model well-known notions of static graphs. All the definitions they give for stream graphs permit to find back these well-known definitions defined for static graphs in case where the stream graph considered is static. For instance, they have defined the degree of a node u in a stream graph as: $d(u) = \sum_{v \in V} |T_{uv}|/|T|$, with $|T_{uv}|$ the duration during which the edge between u and v is present in the stream graph. In the case where the stream graph is a static graph, we obtain effectively the notion of degree defined for static graphs. Indeed, all the edges of a static graph are always present, hence, the adjacent edges of a node u are present during $|T|$. Therefore, for a static stream graph, we have $d(u) = \sum_{v \in V} |T_{uv}|/|T| = n(u)$, where $n(u)$ corresponds to the number of adjacent nodes of u.

Edge-scheduled network. Until now, the models of dynamic graphs presented do not permit to represent the latency, i.e., the time taken (by a message, a robot, *etc.*) to go from one node to another one.

Berman [18] introduces edge-scheduled networks as a multigraph defined by a couple (V, E) where V is a set of nodes and E is a set of arcs. For each arc of an edge-scheduled network, two real weights are assigned: one for the starting time and one for the ending time. For instance, if there is an arc between two nodes u and v such that the starting time is equal to 4 and the ending

time is equal to 7, this means that a communication may start at time 4 between u and v, and if such a communication occurs it will take 3 units of time to reach the node v.

Therefore, this model permits to represent dynamic graphs where arcs may appear and disappear with time, and where the latencies to cross the arcs may also evolve with time.

Time-varying graph. Casteigts et al. [40] model dynamic graphs thanks to time-varying graphs (TVG for short).

A TVG $\mathcal{G}$ is described by a quintuplet $(V, E, \tau, \rho, \zeta)$ where:

- V is a set of nodes.
- $E \subseteq V \times V$ is a set labeled edges. The label of an edge could describe any property. It is possible to have multiple edges between a same pair of nodes as long as the labels of these edges are distinct.
- τ corresponds to the lifetime of the system: $\tau \subseteq T$, where T is the temporal domain of the system. Hence, T is either $\mathbb{N}$ for a discrete time system or $\mathbb{R}^+$ for a continuous time system.
- ρ is a presence function which indicates whether a given edge is available at a given time. More precisely, given an edge e of E and a time t of τ this function returns true if the edge e is present at time t in the dynamic graph, false otherwise.
- ζ is a latency function which gives the time taken to cross a given edge at a given time. More precisely, given an edge e of E and a time t of τ, this function returns a time in T representing the time to cross e at time t.

Note that, we can also say that the TVG $\mathcal{G}$ is described by (G, τ, ρ, ζ), with $G = (V, E)$. In this case, we say that $\mathcal{G}$ is based on G.

The TVG model can be extended by adding a node presence function $\psi : V \times \tau \to \{true, false\}$ that indicates whether a given node is present in the system at a given time, and by adding a node latency function $\phi : V \times \tau \to T$ that indicates the local processing time at a given time.

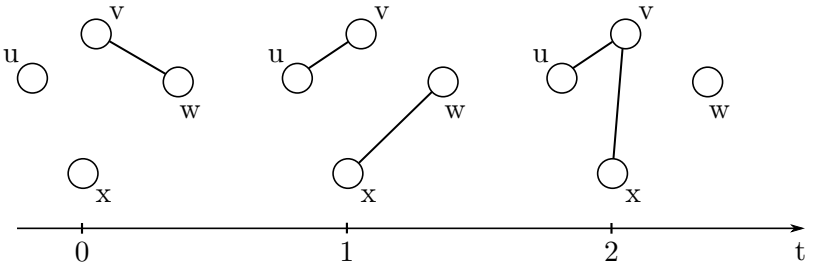
This model, that is similar to the one introduced by Harary and Gupta [101], is very complete. It is richer than all the models of dynamic graphs presented previously. Indeed, it integrates the time to traverse the edges and permits to represent dynamic graphs where edges, nodes and latencies (of nodes and edges) may evolve with time.

Graphical representation. There exist graphical representations for each of the models described previously. Note that, compared to static graphs which are simple to represent, it is hard to represent in a simple way an object that varies with time. Refer to Figure 2.1 for a comparison of the graphical representations of the models of dynamic graphs presented in this section. To compare them, we draw the same dynamic graph in each of the graphical representations of the models described. To be able to draw the same dynamic graph, we have to consider a discrete time system, otherwise we could not be able to consider the evolving graph model or in the edge-scheduled network model.

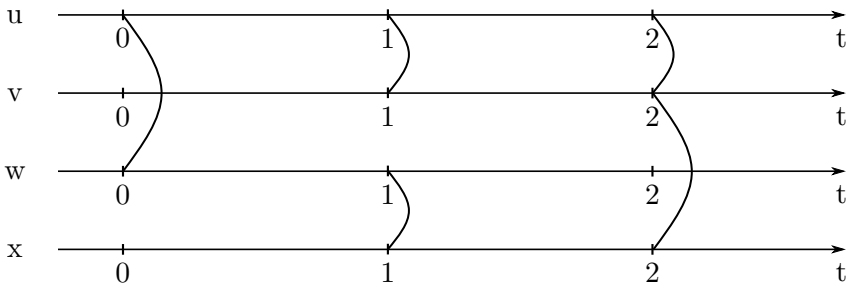
2.1.2 Classification of Casteigts et al. [40, 38]

In addition to the introduction of the TVG formalism, Casteigts et al. [40] have introduced some notions to extend the classical definitions of graph theory. For instance, they have introduced the notion of journey that extends the notion of path. In a static graph, a path is a sequence of edges permitting to link two nodes. In a dynamic graph, a journey is a temporal path: it corresponds to a sequence of edges present at increasing time permitting to link two nodes. For more details, see the formal definition of a journey in Section 2.2 and refer to Figure 2.4.

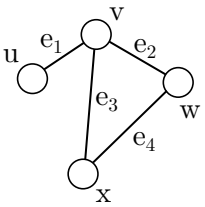
Evolving graph:



Stream graph:

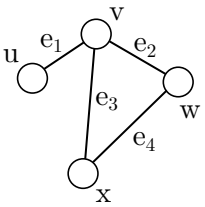


Time-Varying graph:



$\rho(e_1, 1)=\rho(e_1, 2)=1$ $\quad$ $\rho(e_1, 0)=0$
$\rho(e_2, 0)=1$ $\quad$ $\rho(e_2, 1)=\rho(e_2, 2)=0$
$\rho(e_3, 2)=1$ $\quad$ $\rho(e_3, 0)=\rho(e_3, 1)=0$
$\rho(e_4, 1)=1$ $\quad$ $\rho(e_4, 0)=\rho(e_4, 2)=0$

Temporal network:



$\lambda(e_1)=\{1, 2\}$
$\lambda(e_2)=\{0\}$
$\lambda(e_3)=\{2\}$
$\lambda(e_4)=\{1\}$

Edge-scheduled network:

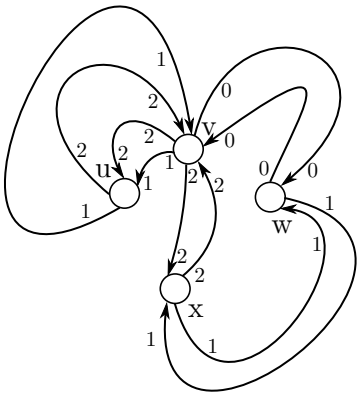


Figure 2.1: Representation of the same dynamic graph in different models of dynamic graphs.

This notion of journey is central since it permits to define the notion of temporal connectivity in dynamic graphs which indicates whether or not the nodes are reachable by the other nodes of the dynamic graph through a journey, and the frequency of this reachability. In other words, like the paths in static graphs that permit to indicate whether a static graph is connected or not, the journeys in dynamic graphs permit to indicate whether a dynamic graph is connected over time or not. Thanks to this notion of journey, Casteigts et al. has classified the different dynamic graphs studied in the state of the art according to their temporal connectivity assumptions. In fact, all the dynamic graphs that satisfy the same temporal connectivity assumptions are regrouped into a same class of dynamic graphs. Casteigts [38] has increased the number of these classes. We present here briefly each of these classes. In the technical parts of this thesis, we only consider a subset of these classes (we will detail them more formally in Section 2.2).

In function of the class of dynamic graphs studied, the problems that may be addressed differ. Indeed, in some dynamic graphs, some problems become impossible to solve. That is why it is important to define multiple classes of dynamic graphs: it permits to find the most dynamic class of dynamic graphs in which a problem may be solved.

Among the classes of dynamic graphs, we can distinguish two groups: one regroups the classes of dynamic graphs having a finite temporal connectivity, a second regroups the classes of dynamic graphs having a recurrent temporal connectivity. It is trivially impossible to solve a recurrent task in a dynamic graph having a finite temporal connectivity. When the temporal connectivity is recurrent, it is not necessary to have the knowledge of the dynamics of the graphs to start the tasks at the appropriate moment to solve them. Indeed, when the temporal connectivity of a dynamic graph is recurrent, the tasks may start at any time, which is not the case when considering a dynamic graph having a finite temporal connectivity in which the tasks should start at the appropriate timing to succeed to be solved.

Classes of dynamic graphs with finite temporal connectivity. We give below the list of the classes of dynamic graphs possessing a finite temporal connectivity.

In the graphs of the class described below, it is possible to perform data aggregations where all the sensors (modeled by the nodes of the graphs) have to send their data to only one sensor.

Temporal sink class: A dynamic graph belongs to the temporal sink class if and only if at least one node can be reached by all the others through a journey.

In each of the graphs of the two classes presented below, it is possible to execute a broadcast from one node u. In the graphs of the first class presented, u reaches each of its destinations in one hop, while in the graphs of the second class the number of hops needed by u to reach each of its destinations may be greater than one.

Temporal star class: A dynamic graph belongs to the temporal star class if and only if at least one node will share an edge at least once with every other node (possibly at different times).

Temporal source class: A dynamic graph belongs to the temporal source class if and only if at least one node can reach all the others through a journey.

Similarly, in the graphs of the two classes below, it is possible to execute a broadcast from any of the nodes of the graph. In the graphs of the first class presented, a node reaches each of its destinations in one hop, while in the graphs of the second class the number of hops for a node to reach each of its destinations may be greater than one.

Temporal clique class: A dynamic graph belongs to the temporal clique class if and only if every pair of nodes will share an edge at least once (possibly at different times).

Temporal connectivity class: A dynamic graph belongs to the temporal connectivity class if and only if every node can reach all the others through a journey.

The last class of this group is a generalization of the temporal connectivity class. Indeed, in the graphs of this class, it is possible to execute a broadcast with feedback from each node.

Round-trip temporal connectivity class: A dynamic graph belongs to the round-trip temporal connectivity class if and only if every node can reach every other node and be reached from that node afterwards.

Classes of dynamic graphs with recurrent temporal connectivity. We give below the list of the classes of dynamic graphs possessing recurrent temporal connectivity.

In each of the dynamic graphs $\mathcal{G}$ (based on G) of the six classes presented below, all the edges of G are infinitely often present in $\mathcal{G}$. In the graphs of the first class, all the nodes can communicate with each other in only one hop, and this at each instant time. In the graphs of the second class, all the nodes can communicate with each other infinitely often in only one hop. In the graphs of the other classes, even if any node can communicate infinitely often with any other one, the number of necessary hops may be greater than one. However, in the graphs of the second, third, and fourth class, the latency of any journey is bounded (which is not the case in the graphs of the last class).

Complete static class: A dynamic graph $\mathcal{G}$ based on G belongs to the complete static class if and only if G is a complete graph, and every edge of G is present in $\mathcal{G}$ at each instant time.

Complete graph of interaction class: A dynamic graph $\mathcal{G}$ based on G belongs to the complete graph of interaction class if and only if G is a complete graph, and every edge of G is present infinitely often in $\mathcal{G}$.

Static class: A dynamic graph $\mathcal{G}$ based on G belongs to the static class if and only if each edge of G is present in $\mathcal{G}$ at each instant time.

Bounded-recurrent-edges class: A dynamic graph $\mathcal{G}$ based on G belongs to the bounded-recurrent-edges class if and only if there exists $\delta \in T$ (where T is either a discrete interval of time included in $\mathbb{N}$ (for a discrete time system), or a continuous interval of time included in $\mathbb{R}^+$ (for a continuous time system)) such that every edge of G is present in $\mathcal{G}$ at least once in any time interval of length δ.

Periodic-edges class: A dynamic graph $\mathcal{G}$ based on G belongs to the periodic-edges class if and only if every edge of G is present in $\mathcal{G}$ periodically.

Recurrent-edges class: A dynamic graph $\mathcal{G}$ based on G belongs to the recurrent-edges class if and only if every edge of G is present in $\mathcal{G}$ infinitely often.

In all the dynamic graphs $\mathcal{G}$ (based on G) of the classes presented afterwards, it may exist eventual missing edge(s): an eventual missing edge is an edge of G such that there exists a time from which this edge is never present in $\mathcal{G}$. The presence of eventual missing edges makes problems harder to be solved. Indeed, without knowledge of the dynamics of a graph, it is not possible to distinguish an eventual missing edge from a missing edge that will appear later in the future. Hence, it is not possible to know if a process has to wait for a missing edge to appear again. Indeed, if a process decides to wait for a missing edge, it may wait indefinitely in the case where this missing edge is in fact an eventual missing edge.

The connected-over-time class is the class with the weakest temporal connectivity assumptions (among the classes of the classification of Casteigts et al.) such that the graphs of this class have a recurrent temporal connectivity. Hence, the graphs of this class are the one with the weakest temporal connectivity assumptions in which it is possible to solve recurrent tasks.

Connected-over-time class: A dynamic graph belongs to the connected-over-time class if and only if there exists infinitely often a journey between any pair of its nodes.

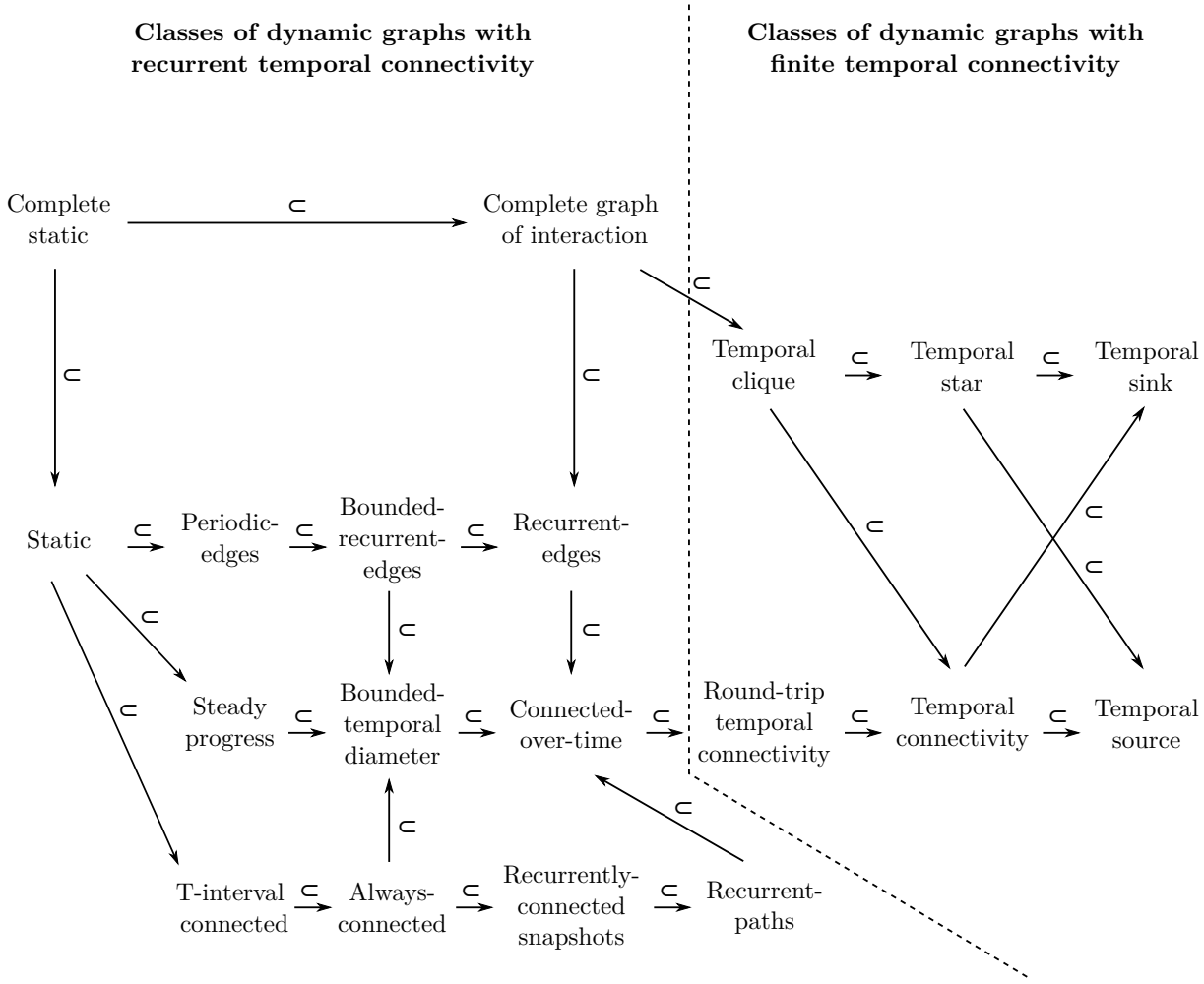


Figure 2.2: Inclusion of the classes of dynamic graphs.

In graphs of the connected-over-time class, there is no assumption on the time of appearance of the journeys. To permit to bound the time of the solutions to problems, we can consider graphs of the bounded-temporal-diameter class in which there is a bound on the time of appearance of the journeys.

Bounded-temporal-diameter class: A dynamic graph belongs to the bounded-temporal-diameter class if and only if there exists $\delta \in T$ (where T is either a discrete interval of time included in $\mathbb{N}$ (for a discrete time system), or a continuous interval of time included in $\mathbb{R}^+$ (for a continuous time system)) such that, for any pair of its nodes, there exists at least one journey every δ units of time.

Having a journey every δ units of time between two nodes does not permit to bound the time to reach each node of the journey. To do so, we can focus on the graphs of the following class.

Steady progress class: A dynamic graph belongs to the steady progress class if and only if at each instant time there is a journey between any pair of its nodes such that each edge of this journey appears in bounded time.

Finally, the graphs of the four last classes that we present are based on the static version of the journey notion (i.e., they are based on the path notion). We present the graphs of these classes

from the ones with the weakest assumptions on the paths to the ones that contain the strongest assumptions on the presence of the paths.

Recurrent-paths class: A dynamic graph belongs to the recurrent-paths class if and only if there exists infinitely often a path between any pair of its nodes.

Recurrently-connected snapshots class: A dynamic graph belongs to the recurrently-connected snapshots class if and only if there exists infinitely often a time at which it is connected.

Always-connected class: A dynamic graph belongs to the always-connected class if and only if at each instant time, it is connected.

T-interval connected class: A dynamic graph belongs to the T-interval connected class if and only if it is connected at each instant time and for all time interval I of length T, there exists a spanning connected subgraph of G whose edges are present at any time of I.

Inclusion. Due to their definitions, there exists multiple inclusion relations between all the classes presented (refer to Figure 2.2). Note that the relation of inclusion between any pair of classes is a strict inclusion. This hierarchy of class has deep consequences on calculability, refer to Section 4.1 for more details.

2.2 Model

We define formally here all the notions and assumptions related to dynamic graphs used afterwards.

Main notions (evolving graph, characteristic graphs, and edges). In the following, we give the definitions of evolving graph, and of the characteristic graphs and edges we have to deal with when considering dynamic graphs.

We consider the time as discretized and mapped to $\mathbb{N}$. We study dynamic graphs such that the set of nodes is static, only the edges may appear and disappear with time. The most suitable formalism is then the one of *evolving graph* introduced by Ferreira [80]. More formally, we can define an evolving graph as follows.

Definition 2.1 (Evolving graph)**.** *An evolving graph $\mathcal{G}$ is an ordered sequence $(G_0, G_1, G_2, \ldots)$ of subgraphs of a given static graph $G = (V, E)$ with V a set of nodes and E a set of edges. $\mathcal{G}$ is said to be based on G, and G is called the footprint of $\mathcal{G}$.*

For any $i \geq 0$, we call $G_i = (V, E_i)$ the snapshot of $\mathcal{G}$ at time i. We say that the edges of $E_i \subseteq E$ are present *in $\mathcal{G}$ at time i. As said, V is a static set and we denote $|V|$ by n.*

The footprint describes the set of edges that are allowed to be present in $\mathcal{G}$. While the footprint gives the allowed connections, the underlying graph, defined formally below and denoted $U_{\mathcal{G}}$, captures the actual connections in $\mathcal{G}$ (that is, the set of edges that are present at least once in the lifetime of the graph). Since some edges of G may never be present in $\mathcal{G}$, these edges are also not present in $U_{\mathcal{G}}$.

Definition 2.2 (Underlying graph of an evolving graph $\mathcal{G}$)**.** *The underlying graph of an evolving graph $\mathcal{G} = ((V, E_0), (V, E_1), (V, E_2), \ldots)$ is the static graph $U_{\mathcal{G}}$ such that $U_{\mathcal{G}} = (V, E_{\mathcal{G}})$ with $E_{\mathcal{G}} = \bigcup_{i \in \mathbb{N}} E_i$.*

Even if $U_{\mathcal{G}}$ represents the actual connections in $\mathcal{G}$, some edges of $U_{\mathcal{G}}$ may stop to be present in $\mathcal{G}$ after a certain time. To capture the connections that are actually usable infinitely often we

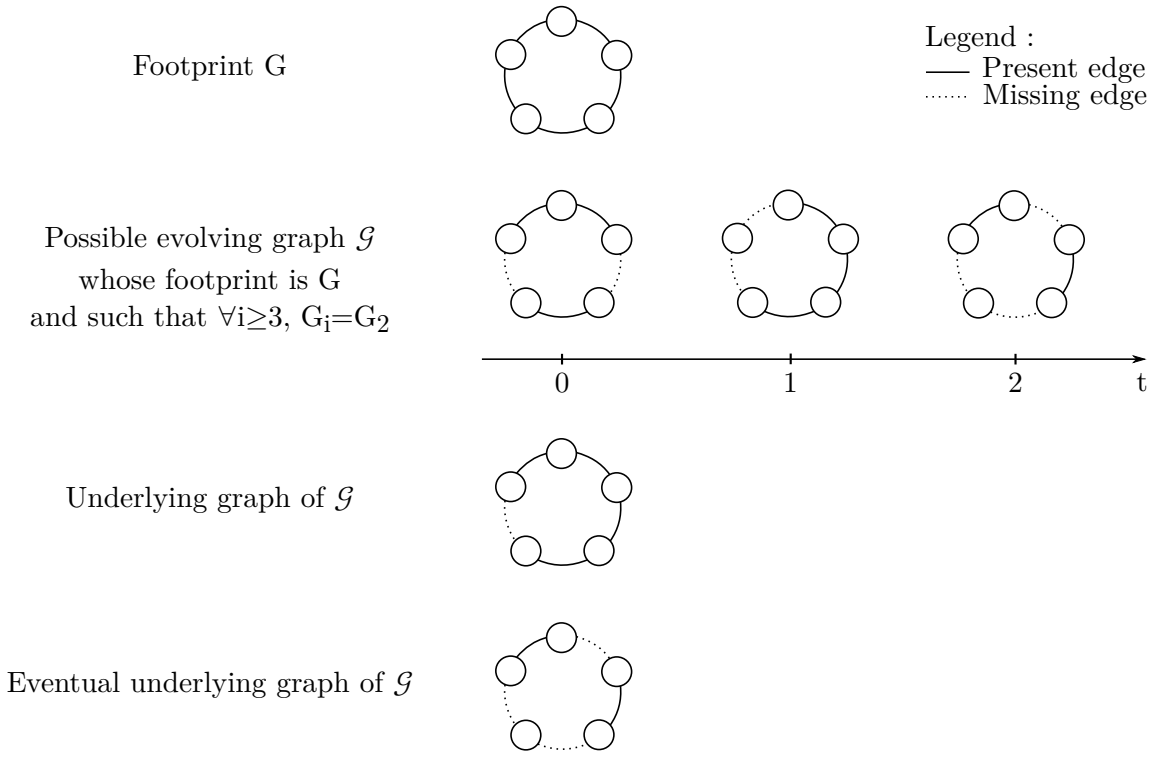


Figure 2.3: Evolving graph and characteristic graphs.

consider the eventual underlying graph. To define formally what the eventual underlying graph is, we need to define some characteristic edges.

An *eventual missing edge* is an edge of E such that there exists a time after which this edge is never present in $\mathcal{G}$. These kinds of edges cannot be crossed after a certain time, they thus are not present in the eventual underlying graph. By opposition, a *recurrent edge* is an edge of E that is not eventually missing.

The *eventual underlying graph*, denoted $U^\omega_\mathcal{G}$, is then defined as follows.

Definition 2.3 (Eventual underlying graph of an evolving graph $\mathcal{G}$)**.** *The eventual underlying graph of an evolving graph $\mathcal{G} = ((V, E_0), (V, E_1), (V, E_2), \ldots)$ is the static graph $U^\omega_\mathcal{G}$ such that $U^\omega_\mathcal{G} = (V, E^\omega_\mathcal{G})$ where $E^\omega_\mathcal{G} = \bigcap_{i\in\mathbb{N}}(\bigcup_{j\geq i} E_j)$.*

An example illustrating the notions presented in this paragraph can be found in Figure 2.3.

Classes of dynamic graphs. In Section 2.1.2, we presented multiple classes of dynamic graphs. We define formally in this paragraph the classes of dynamic graphs we focus on in this thesis (see [40]). By abuse of language, in the remainder of this thesis, we call an "X graph" a graph that belongs to the class X.

The class $\mathcal{ST}$ (static), defined formally below, is the class with the weakest dynamics we consider in this thesis in the sense that the connectivity assumptions of the graphs of this class are the strongest possible.

Definition 2.4 (Static graph ($\mathcal{ST}$ graph))**.** *An evolving graph $\mathcal{G} = (G_0, G_1, G_2, \ldots)$ is a $\mathcal{ST}$ graph if and only if for any $i \in \mathbb{N}$, G_i is equal to the footprint G.*

In this thesis, we also consider the class $\mathcal{BRE}$ (bounded-recurrent-edges) [39], which is defined below.

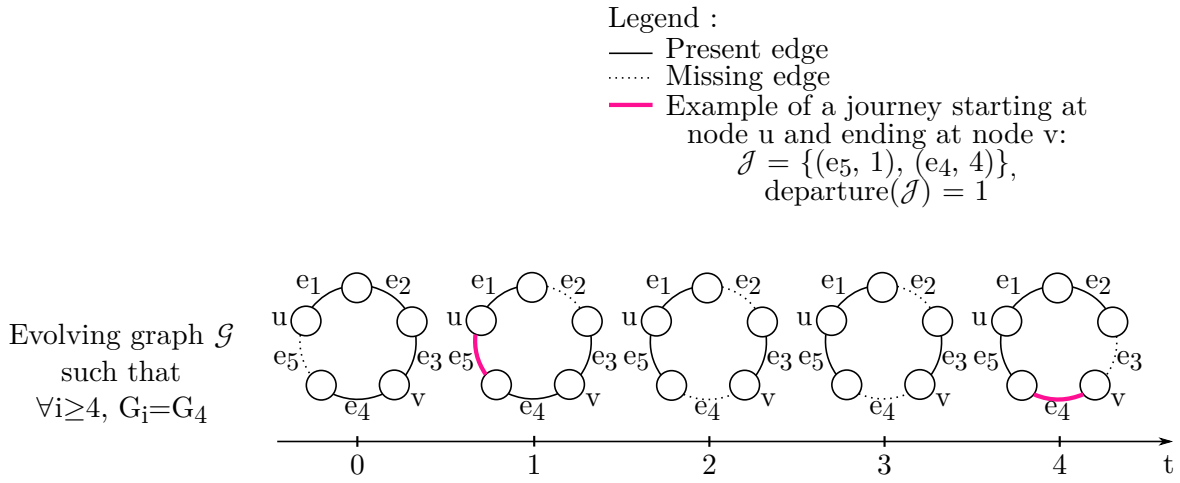


Figure 2.4: Example of a journey starting at node u and ending at node v.

Definition 2.5 (Bounded-recurrent-edges graph ($\mathcal{BRE}$ graph))**.** *An evolving graph $\mathcal{G}$ is a $\mathcal{BRE}$ graph if and only if there exists a $\delta \in \mathbb{N}$ such that each edge of the footprint G is present in $\mathcal{G}$ at least once in any time interval of length δ.*

Now, we present the class $\mathcal{RE}$ (recurrent-edges) [106, 39].

Definition 2.6 (Recurrent-edges graph ($\mathcal{RE}$ graph))**.** *An evolving graph $\mathcal{G}$ is a $\mathcal{RE}$ graph if and only if each edge of the footprint G is a recurrent edge in $\mathcal{G}$.*

To sum up, in the graphs of the three classes presented above, there does not exist eventual missing edge. All the edges of a $\mathcal{ST}$, $\mathcal{BRE}$ or $\mathcal{RE}$ graph are recurrent. In other words, if an evolving graph $\mathcal{G}$ is a $\mathcal{ST}$, $\mathcal{BRE}$ or $\mathcal{RE}$ graph then its eventual underlying graph $U^{\omega}_{\mathcal{G}}$ is equal to its footprint G.

We now present the two last classes we consider in this thesis. In the graphs of these two classes, it is possible to have eventual missing edges.

First we consider the class $\mathcal{AC}$ (always-connected) [127, 116].

Definition 2.7 (Always-connected graph ($\mathcal{AC}$ graph))**.** *An evolving graph $\mathcal{G} = (G_0, G_1, G_2, \ldots)$ is a $\mathcal{AC}$ graph if and only if for any $i \in \mathbb{N}$, G_i is connected.*

Below, we define the class with the strongest dynamics we consider in this thesis. In the graphs of this class, the edges may appear and disappear unpredictably without any recurrence, periodicity, or stability assumption. The only assumption made in the graphs of this class is that each node is infinitely often reachable from another one through a journey.

To define formally the class $\mathcal{COT}$, we first need to define formally a journey.

Definition 2.8 (Journey)**.** *A journey $\mathcal{J}$ in an evolving graph $\mathcal{G}$ is a sequence of couples $((e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k))$ such that $(e_1, e_2, \ldots, e_k)$ is a path in G, for any i such that $0 < i < k, t_{i+1} > t_i$, and for any j such that $j \leq k$, the edge e_j belongs to E_{t_j}. The time t_1 is denoted departure($\mathcal{J}$) and corresponds to the time at which the journey starts.*

Figure 2.4 pictured an example of a journey starting at node u and ending at node v.

We can now formally define the class $\mathcal{COT}$.

Definition 2.9 (Connected-over-time graph ($\mathcal{COT}$ graph))**.** *An evolving graph $\mathcal{G} = ((V, E_0), (V, E_1), (V, E_2), \ldots)$ is in $\mathcal{COT}$ if and only if for any pair of nodes $(u, v) \in V^2$ and for any time $t \in \mathbb{N}$ there exists a journey $\mathcal{J}$ starting at node u and ending at node v such that departure($\mathcal{J}$) $> t$.*

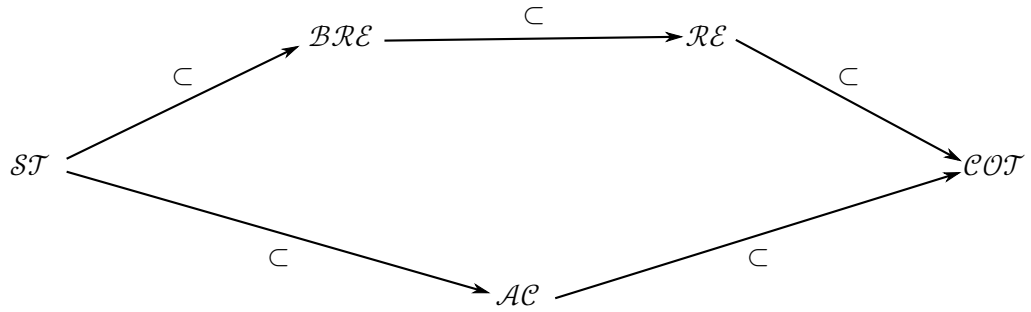


Figure 2.5: Inclusion of the classes of dynamic graphs.

In other words, a $\mathcal{COT}$ graph is a dynamic graph in which there exists infinitely often a journey between any pair of nodes. This implies that the class $\mathcal{COT}$ is the set of all evolving graphs such that their eventual underlying graph is connected [78].

As indicated in Section 2.1.2, there exists an inclusion relation between all the classes presented: $\mathcal{ST} \subset \mathcal{BRE} \subset \mathcal{RE} \subset \mathcal{COT}$ and $\mathcal{ST} \subset \mathcal{AC} \subset \mathcal{COT}$ (refer to Figure 2.5).

Thesis assumptions. In this thesis, we restrict ourselves to evolving graphs whose footprints are anonymous (i.e., the nodes cannot be distinguished), undirected (i.e., the edges are bidirectional), and simple graphs. We mostly consider graphs whose footprints are rings, except in Section 11.1 where we consider more general graphs. Although the rings we considered are undirected, to simplify the presentation and discussion, we, as external observers, distinguish between the clockwise and the counter-clockwise (global) direction in the ring.

Operations on evolving graphs. For the sake of some proofs in this thesis, we need to introduce two operators on evolving graphs.

The first one, denoted $\setminus$, removes some edges of an evolving graph for some time ranges. More formally, from an evolving graph $\mathcal{G} = ((V, E_0), (V, E_1), (V, E_2), \ldots)$, we define the evolving graph $\mathcal{G}\setminus\{(\tilde{E}_1, \tau_1), \ldots, (\tilde{E}_k, \tau_k)\}$ (with for any $i \in \{1, \ldots, k\}$, $\tilde{E}_i \subseteq E$ and $\tau_i \subseteq \mathbb{N}$) as the evolving graph $\{(V, E'_0), (V, E'_1), (V, E'_2), \ldots\}$ such that: $\forall t \in \mathbb{N}, \forall e \in E, e \in E'_t \Leftrightarrow e \in E_t \wedge (\forall i \in \{1, \ldots, k\}, e \notin \tilde{E}_i \vee t \notin \tau_i)$.

An example illustrating this operation is presented in Figure 2.6.

The second operator, denoted $\otimes$, concatenates a prefix of an evolving graph with a suffix of another one. The two evolving graphs concatenated are defined on the same set of nodes. Formally, given two evolving graphs, $\mathcal{G} = (G_0, G_1, G_2, \ldots)$, with for any $i \in \mathbb{N}$, $G_i = (V, E_i)$, and $\mathcal{H} = (H_0, H_1, H_2, \ldots)$, with for any $i \in \mathbb{N}$, $H_i = (V, E'_i)$ and an integer t, the evolving graph $\mathcal{G} \otimes_t \mathcal{H}$ is the evolving graph $\mathcal{G}' = (G'_0, G'_1, G'_2, \ldots)$, with for any $i \in \mathbb{N}$, $G'_i = (V, E_i")$, defined by: $e \in E_i"$ if and only if $i \leq t \wedge e \in E_i$ or $i > t \wedge e \in E'_i$.

An example illustrating this operation is presented in Figure 2.7.

Other definitions/properties. Except if explicitly written otherwise, we define the distance between two nodes u and v of $\mathcal{G}$ by the length of a shortest path between u and v in its footprint G.

We say that two evolving graphs $\mathcal{G}$ and $\mathcal{G}'$ share the same prefix if there exists a time t such that for any $i \leq t$, $G_i = G'_i$.

We say that a node u satisfies the property $OneEdge(u, t, t')$ if and only if an adjacent edge of u is continuously missing from time t to time t' while the other adjacent edge of u is continuously present from time t to time t'.

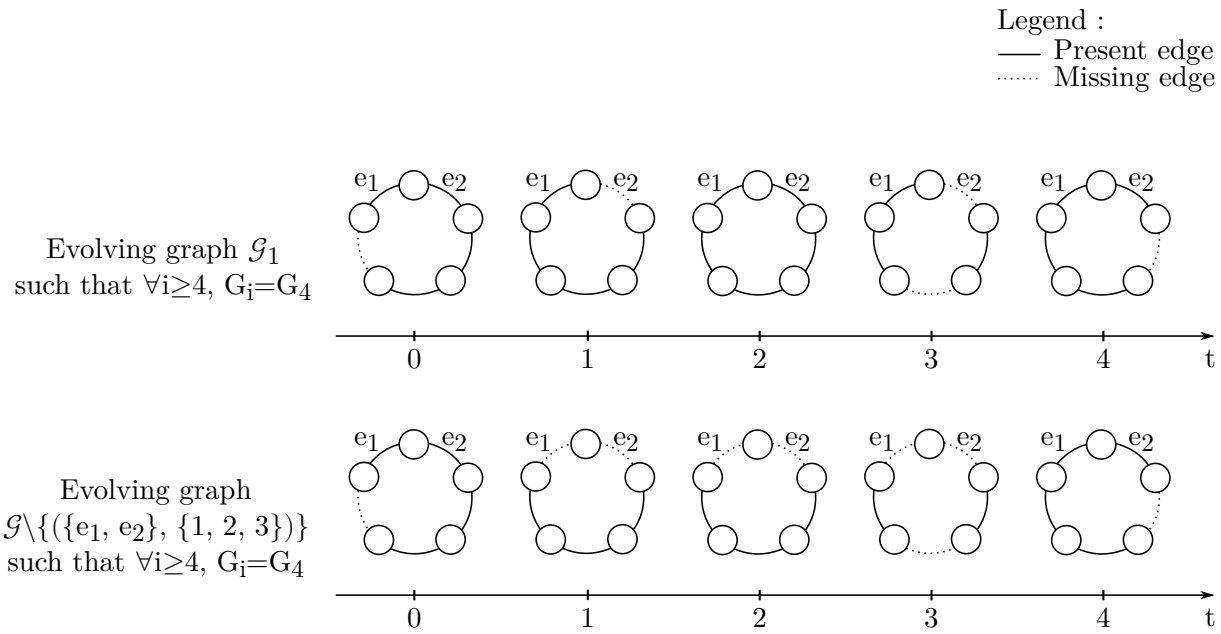


Figure 2.6: Illustration of the $\setminus$ operator.

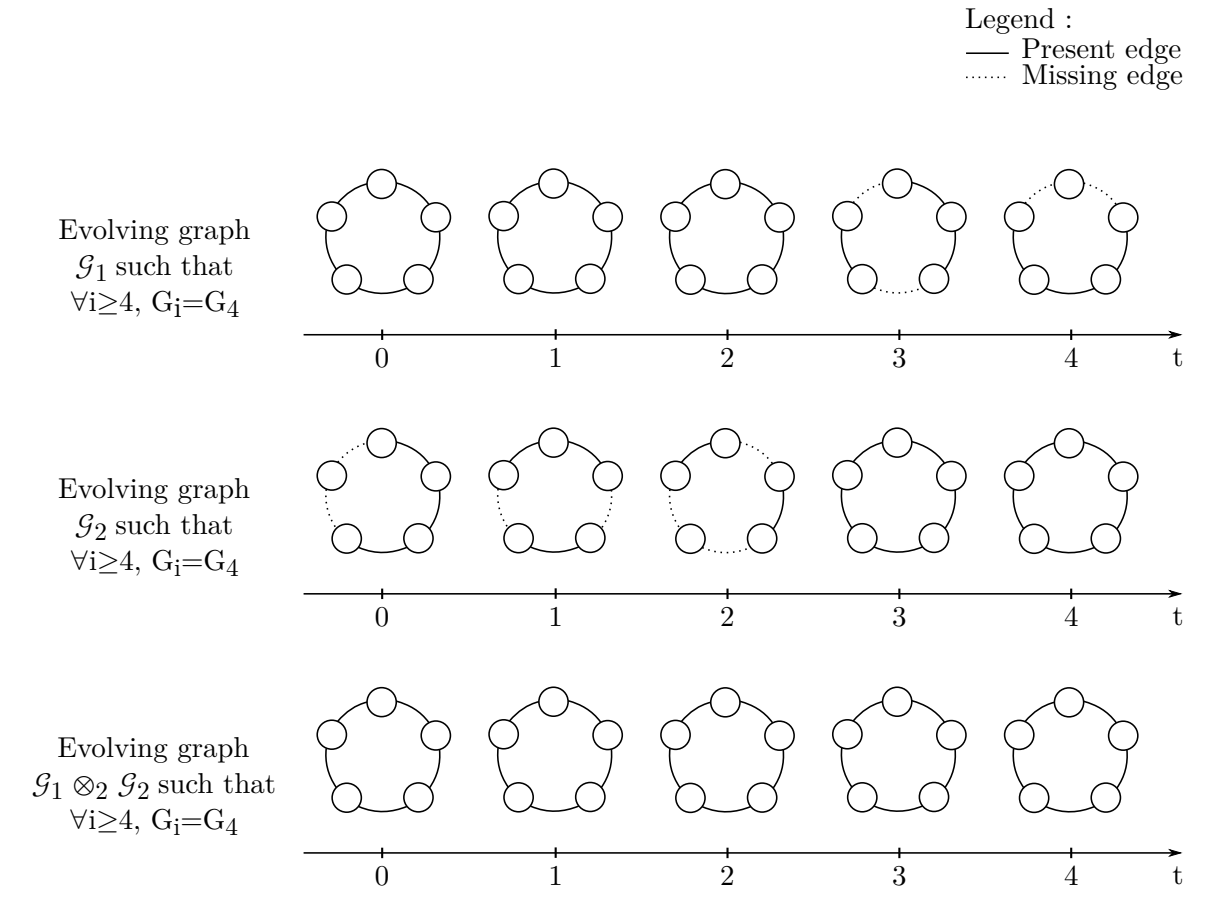


Figure 2.7: Illustration of the $\otimes$ operator.

Chapter 3

Robots

Contents

In this thesis, we are interested in the study of distributed systems which are systems composed of multiple entities. The entities of a distributed system collaborate to solve a same task. They are autonomous: they take their own decisions without the help of a central entity. These decisions are taken based on their own vision of the system. More specifically, a decision is taken thanks to the execution of an algorithm (i.e., sequence of ordered instructions). The set of all the algorithms of all the entities of a distributed system is a distributed algorithm. This is thanks to the execution of a distributed algorithm that the entities of a distributed system collaborate to solve tasks.

It is hard to solve problems using distributed systems. Indeed, in a distributed system all the entities have only a local vision of the system whereas the entity of a central system has a global vision of the system. Moreover, in a distributed system the entities have to collaborate to solve a task, which is not the case in a central system in which only a single entity is present. However, using distributed systems have some advantages: if one or multiple entities fail, it may be still possible to solve the studied task, whereas if the entity of a central system fails, it is not possible to solve the task anymore. Moreover, using multiple entities may permit to solve some tasks more efficiently than solving them with only a single entity: the entities of a distributed system may distribute the task to solve between themselves.

In this context, we are more precisely interested in the study of swarms of robots. A swarm of robots is a distributed system whose autonomous entities are robots. Each robot is able to take decisions without the control of a central authority. Each robot is endowed with motion actuators, i.e., it is able to move in the environment in which it evolves. Robots are also endowed with sensing captors permitting them to sense their environment. Robots are a theoretical model in the sense that they represent assumptions that are made on the processes of a distributed system (such that these processes have the ability to move in their environment). The goal of the study of theoretical distributed systems is to determine the minimum assumptions made on the processes of a distributed system to solve a task; and to find distributed algorithms permitting to solve this task in a system with such assumptions.

With the increase of drones, automatic vehicles, *etc.* (that can be modeled thanks to swarms of robots), robots are more and more used. Using robots is interesting, specifically in the case where the actions to perform are dangerous for humans (for instance the exploration of a dangerous

zone) or the tasks to execute are too hard for humans (for instance the transportation of a heavy load).

The tasks solvable by the robots depend on their capacities (Can they see the whole environment in which they evolve?, Can they orient themselves in their environment? ...).

Every system being subject to faults, in order to tolerate faults to which robots may be subject, it is better to use multiple robots with few capacities rather than only one robot endowed with lots of capacities. Indeed, the use of only one powerful robot is risky since if only one of its capacity breaks, the robot may be unable to solve the task it has to perform. Moreover these kinds of powerful robots are expensive. Using multiple robots with few capacities that cooperate in order to solve a task is, however, convenient since it is less probable that the robots would be subject to faults (the less robots have capacities, the less they will be prone to faults). Moreover these robots are less expensive.

Considering swarms of robots with few capacities, some articles, initiated by Suzuki and Yamashita [136], analyze the necessary and sufficient conditions on the capacities of the robots or on the number of the robots in order to solve some important tasks like the exploration or the gathering in a deterministic way.

In the continuation of these work [136, 137, 92], we study swarms of robots that cooperate in order to solve a common task. As indicated, swarms of robots are theoretical distributed systems: they model realistic systems but sometimes in a simpler way. Indeed, in these systems it is possible to assume that the entities have infinite memory or infinite computation capabilities.

In this chapter, we first give an overview of the state of the art on swarms of robots (see Section 3.1), then we describe the formal model we use all along this thesis (see Section 3.2).

3.1 State of the Art

In this section, we present the state of the art about robots. On this purpose, we first describe, in Section 3.1.1, the computational model of the robots, then we present, in Section 3.1.2, the different environments in which the robots may evolve. We then describe, in Section 3.1.3, the main assumptions considered on robots in the literature. Finally, we give, in Section 3.1.4, the classical problems solved using robots.

3.1.1 Computational Model

In this section, we present the computational model of the robots. This computational model, explained below, is based on Look-Compute-Move cycles (denoted L-C-M cycles). It has been introduced by Suzuki and Yamashita [136] and is now used in most of the articles dealing with swarms of robots.

When robots execute algorithms, they execute an infinite number of L-C-M cycles. Each cycle is composed of a Look, a Compute and a Move phase described as follows. During the Look phase, a robot uses its sensing captors in order to observe its environment. Based on these observations, during the Compute phase a robot computes (thanks to its algorithm) a destination towards which it wants to move. Then, during the Move phase it effectively moves to the destination it computed during the Compute phase.

There exist three variants of this model depending on the synchronization of the phases of the L-C-M cycles of the robots. We describe below each of these variants.

FSYNC: Suzuki and Yamashita [137] introduced the *FSYNC* (for *fully-synchronous*) model: in the FSYNC model the time is discrete, and at each instant time all the robots of the system execute atomically and simultaneously their L-C-M cycle. Note that a unit of time is also called a *round*, therefore in this model each robot execute a L-C-M cycle at each round.

SSYNC: Suzuki and Yamashita [136] introduced the ATOM model, also known as the SYm or *SSYNC* (for *semi-synchronous*) model, in which the time is discrete, and, at each instant time, a non-empty subset of the robots of the system execute atomically and simultaneously their L-C-M cycle. This model is a generalization of the FSYNC model.

ASYNC: In the CORDA model [92], also known as *ASYNC* (for *asynchronous*) model, the robots execute their L-C-M cycles independently. This is done by adding a wait phase at the beginning of each L-C-M cycle of the robots during which the robots just stop their execution. In the ASYNC model, the Wait, Compute, and Move phases of the robots are of arbitrary but finite length and may be different for each cycle and for each robot. Only the Look phase is supposed to be instantaneous. Besides, there is an arbitrary but finite time between each phase of the same cycle. Therefore, in this model it is possible for a robot to see during its Look phase robots while they are moving. This model, introduced by Flocchini et al. [92], is the more realistic one compared to the FSYNC and SSYNC models. However since the ASYNC model encompasses the two other models (i.e., every execution allowed in the FSYNC and the SSYNC models are also allowed in the ASYNC model, but the reverse is not true), solving problems in this model is harder than in the FSYNC or the SSYNC models.

3.1.2 Environments

The first work considering swarms of robots [136, 137] consider that robots evolve in the plane: robots are able to move in the whole two-dimensional Euclidean space. Then, in order to constrain the possible locations where the robots may be, some work consider discrete static environments modeled by graphs [113, 75]. In these graphs, the nodes represent the locations of the plane where the robots may be, and the edges represent the paths to go from one location to another one. More recently, graphs are used to represent dynamic environments. In these environments the nodes and edges may appear and disappear with time. These dynamic graphs are useful to represent unstable environments that may change over time, like for instance, a transportation network, a building in which doors are closed and open over time, or streets that are closed over time due to work in process or traffic jam in a town.

In this section, we first present the literature dealing with robots in continuous space, then we present the state of the art about robots evolving in discrete static environment, and finally, we describe the literature about robots evolving in discrete dynamic environment.

Continuous space. In the literature, robots may evolve in the continuous space. While considering the continuous space, some articles deal with robots evolving in a line (one-dimensional Euclidean space) [45, 37], some others study robots evolving in the plane (two-dimensional Euclidean space) [72, 36, 43, 50, 56, 4], some focus on robots evolving in the three-dimensional Euclidean space [139, 143, 121]. More recently an article deals with robots evolving in N dimensional Euclidean space, with $N \in \mathbb{N}$ [123].

When evolving in the continuous space, robots are endowed with captors with an infinite precision: during the Look phase, the robots are able to see the points of the continuous space where the other robots of the system are located. Moreover, to move in the continuous space, the robots need to have an infinite precision of computation (to designate the point in the continuous space they want to reach). Hence, during the Compute phase, robots are able to compute with infinite precision. During the Move phase, they move in straight line to the destination point they computed during the Compute phase. The movements of the robots are said to be rigid if the robots succeed to reach, during their Move phase, the destination point they have computed during their previous phase [95, 143, 6]. In some models, the Move phase of a robot is aborted before it reaches its destination point [95, 139, 102]. More precisely, there exists a value $\delta > 0$

such that if the destination point is at distance less than δ, then the robot reaches its destination point; otherwise it moves of δ units of length towards its destination point.

Moreover, while considering the continuous space, robots are generally dimensionless, i.e., viewed as points in this space [72, 36, 43]. However, there exists some articles where the robots are considered to be fat, i.e., robots are viewed as disks [50, 56, 4].

Discrete static environment. In order to restrict the computational precision of the robots and the locations where the robots may be, some articles study robots that evolve in discrete static environments, i.e., static graphs. In these graphs, the nodes represent the locations where the robots may be located, and the edges represent the possibilities for a robot to move from one location to another one.

Generally, each robot is able to cross at most one edge at each L-C-M cycle. Most of the articles consider undirected graphs [113, 85]: in these graphs the edges are bidirectional. However, it exists some articles dealing with directed graphs [17, 96] in which edges can be crossed only in one direction. Generally, the graphs in which robots evolve are composed of anonymous nodes [64, 55], i.e., robots are not able to distinguish the nodes.

While considering discrete static graphs, multiple topologies are studied: chains [85], rings [83, 55, 21], trees [84, 135], grids [63, 81], multidimensional grids [13], tori [64] and arbitrary graphs [69, 51].

In the literature, many work consider ring-shaped graphs. These graphs are the simplest ones with symmetry. Most of the time the difficulty of problems comes from this topology and the idea of the solution in ring-shaped graphs can be extended to more general graphs.

Discrete dynamic environment. More recently, robots are considered to evolve in discrete dynamic environments, i.e., dynamic graphs. As indicated in Section 2.1, dynamic graphs permit to model unstable systems such as wireless networks, environments in which a natural disaster has occurred, *etc.* In the literature, the dynamic graphs in which robots evolve are graphs such that the set of nodes is fixed, but the edges may appear and disappear with time.

Similarly as in static graphs, the nodes of the graphs represent the locations where the robots may be located and the presence of an edge at a given time represents the possibility for a robot to move from one location to another one at this time.

In all the articles of the state of the art dealing with robots evolving in dynamic graphs, each robot crosses at most one edge at each L-C-M cycle. Some articles study undirected dynamic graphs where the edges are bidirectional [142, 3], while some others focus on directed dynamic graphs where the edges are directed (i.e., edges can be crossed only in one direction) [105, 90]. Some articles deal with anonymous nodes [142, 3], while some others consider non-anonymous nodes. Among the last category of articles, some consider that nodes of the graphs have distinct identifiers [105, 90], some others just consider that some nodes are identically marked [122], while some others assume that there is one marked node (i.e., node that is distinguishable from the other nodes) [125].

Most of the articles dealing with robots in dynamic graphs consider ring-shaped graphs [110, 142, 106, 125, 122, 3, 128], however, some study arbitrary graphs [105, 90], while some others focus on specific graphs like cactus (i.e., tree of cycles) graphs [104] or torus graphs [99].

Multiple kinds of dynamics are considered. Indeed, some articles consider $\mathcal{AC}$ graphs (graphs connected at each instant time) [106, 104, 125, 122, 99]. Some articles study graphs in which all the edges appear infinitely often, with [106] or without [110] a bound on their time of appearance, while some others consider dynamic graphs where the edges appear periodically [105, 90]. There is a work that focuses on dynamic graphs in which the edges appear in a random way [142]. Some authors focus on the most dynamic graphs (of the classification of Casteigts et al. [40, 38]) possessing a recurrent temporal connectivity, i.e., they focus on $\mathcal{COT}$ graphs [128]. Finally, some authors introduce another kind of dynamics, called vertex permutation, such that the topology of

the graph at each instant time is maintained but the edges between the (fixed set of) nodes may change [3]. For instance, at each instant time, the topology is a ring, but at a time t the node u may be adjacent to a node v and a node w and the next instant time the node u may be adjacent to two different nodes. Agarwalla et al. [3] also study $\mathcal{AC}$ graphs with vertex permutation. For the case of the ring, this implies that at each instant time the topology of the graph is either a ring or a chain, and at each instant time the connections between the nodes may change.

Refer to Section 5.2 and Section 8.2 for more details about existing work studying robots in dynamic graphs.

3.1.3 Robot Capacities

The robots may be endowed with multiple capacities that represent the assumptions made on the processes of a distributed system. The capacities capture the abilities delivered by the captors possessed by the robots. They indicate whether the robots have memory or not, whether they are endowed with compasses or not, *etc.* If the robots are endowed with many capacities, then they are more subject to faults since each capacity involves a mechanism that may be broken. Therefore, the goal when studying robots is to use robots with the fewer capacities possible. However, it is not always possible to solve problems with robots devoid of capacity. Hence, generally, the articles of the state of the art focus on the feasibility of problems depending on the capacities of the robots by proving the necessity and/or the sufficiency of each of them. In this section, we detailed the most widespread capacities of the robots.

Uniformity: The robots may all execute the same algorithm, in this case they are said to be uniform [21, 125, 81]. However, it is also possible for the robots not to possess the same algorithm.

Identification: It is possible to consider anonymous robots [21, 81, 122], in this case they are indistinguishable from each other. It is also possible to consider robots with identifiers [115, 62, 3]. Generally, the robots possess distinct identifiers, and each robot knows, at least initially, only its own identifier. Having distinct identifiers permit, for instance, to elect a leader (i.e., a robot that is distinguished by all the robots of the system) that will lead the coordination of the robots [36]. Some authors consider also that robots may be endowed with identifiers that are not necessarily distinct [32]: when two robots share the same identifier they are said to be homonyms.

Visibility range: Robots are endowed with visibility sensors that permit them to sense their environment. More precisely they are able to see the other robots of the system. In some articles, the robots are able to see all the robots of the environment (either continuous space [36, 72] or discrete environment [63, 85]), in this case the robots have unlimited visibility; otherwise they have limited visibility. When the robots have limited visibility, they are called myopic robots. Myopic robots in the plane only see robots located in their radius of visibility [93, 134]. A myopic robot in a graph that sees at distance x means that it sees all the robots of the environment that are located to at most x hops away from its current position [55, 100]. When myopic robots see only their own node, we say that the robots have local vision, by opposition to robots with global vision that have unlimited visibility. In the plane, there also exist some articles where the view of robots may be obstructed, either by other robots located in their sight [50, 48] or by obstacles on the plane [124].

Multiplicity detection: There exists two kinds of multiplicity detections: when a robot is endowed with the *weak multiplicity detection* and moves in a continuous space [92] (resp. a discrete environment [85]) it is able to see if there is none, one or multiple robots on each point (resp. on each node) of its visibility range, but it is not able to know the exact number of robots located on each point (resp. on each node); when a robot is endowed with *strong*

multiplicity detection and moves in a continuous space [71] (resp. a discrete environment [63]) it is able to see the exact number of robots that are located on each point (resp. on each node) of its visibility range.

Memory: Robots may be oblivious, i.e., do not have persistent memory [113, 83, 55]. More precisely, oblivious robots have their variables that are reset at the end of each L-C-M cycle. However, during a L-C-M cycle an oblivious robot may use an infinite amount of memory to compute. Conversely, robots may possess persistent memory to store variables [51, 69, 125]. The values of these variables subsist during the different L-C-M cycles: they are not reset at the end of each L-C-M cycle.

Orientation and length units: When moving in the plane, robots may possess a compass that indicates them the cardinal directions. Robots that possess compasses agree then on the direction and orientation of the two axes [47, 67]. There also exist partial compasses that permit robots to agree on the direction and orientation of only one axis [19]. Two robots have the same chirality [143] if it is possible to obtain the direction and orientation of the two axes of one of them thanks to a rotation of the direction and orientation of the two axes of the other one. Refer to Figure 3.1 for an illustration on the orientation capacity of robots moving in the plane. Independently of this orientation capacity, the robots may have their own unit of length [43, 94], or they can share the same unit of length [47, 67]. When moving on graphs, the term *chirality* is also used: if the robots have the same chirality [12, 47] this implies that they agree on the labeling of the ports of the nodes of the graph, otherwise they have not the same chirality [21, 65]. Refer to Figure 3.2 for an illustration of the chirality in graphs.

Communication: When robots communicate only in an implicit way by observing the positions of the other robots of the system, they are said to be silent [42, 21, 63]. Robots may also communicate in an implicit way by encoding information thanks to their movements and by observing these movements, such robots are called stigmergic robots [66]. It is also possible for the robots to be able to communicate explicitly. Indeed, robots may exchange values of their variables when located on a same point of the continuous space or on a same node of the graph [54, 53, 26]. They are also able to communicate explicitly with all the robots of the system in models where they are endowed with lights [130, 52, 102, 6]: lights encode information and are visible by all the robots of the system, they hence permit an explicit communication between the robots. Note that, the lights also represent a part of the persistent memory of the robots. Similarly, while evolving in discrete (either static or dynamic) environment, it may exist whiteboards on nodes that can be read and written by the robots located on them [110, 88, 128]: these whiteboards permit the robots to communicate in an explicit way, and constitute memories located on nodes.

3.1.4 Classical Problems

There exist multiple problems that may be solved thanks to robots. Robots may solve very complex tasks like rescuing a person in danger. All these complex tasks can be decomposed in multiple basic tasks. From an algorithmic point of view, we focus only on these elementary tasks. We present in this section the classical elementary tasks solved thanks to robots depending on the environment in which they evolve.

Continuous space. The *gathering* problem is a fundamental task in which robots, initially scattered in the environment, have to be located on one point, not known in advance, of the continuous space in finite time [43, 93, 50]. When only two robots have to gather, the gathering problem is called the *rendezvous* problem [109, 95]. Since the robots agree on a point of the continuous space, and meet themselves on this point, the gathering problem is related to the

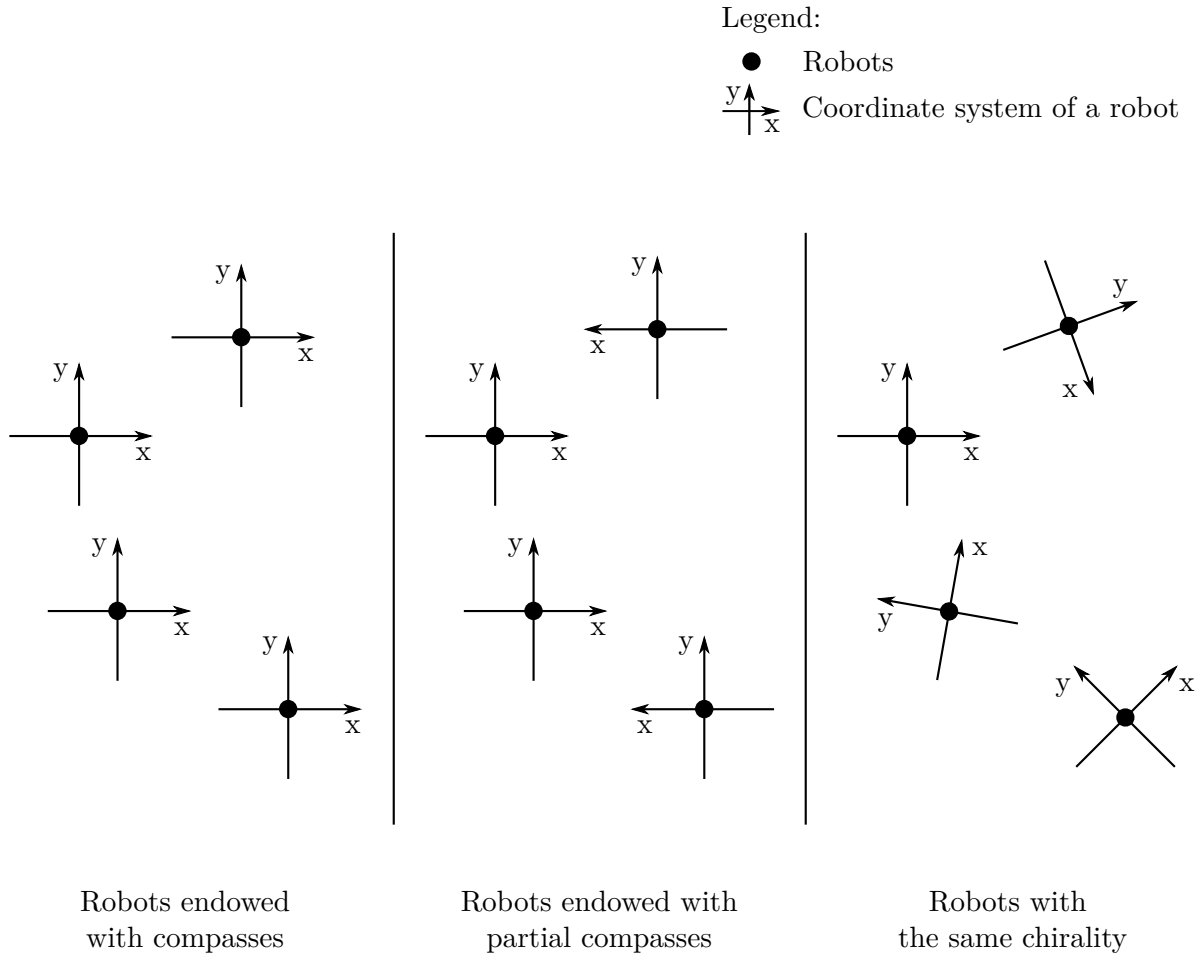


Figure 3.1: Illustration of the orientation capacity of robots evolving in the plane.

consensus problem in classical distributed systems [57, 107]. The gathering problem is a basic problem of the literature. Indeed, once the gathering problem is solved, the gathered robots may exchange information or divide some tasks between them. Note that some solutions only succeed to converge through the gathering: the set of robots, with arbitrary initial positions, asymptotically approach the exact same, but unknown beforehand, position [7, 111]. Such solutions cannot be composed with other algorithms to solve more complex tasks since they only converge through the gathering and hence never terminate.

The *approach* problem has some familiarity with the rendezvous problem: to solve the approach problem, in finite time, the robots (that possess a limited visibility range) have to be in the range of vision of each other [51, 67].

At the opposite of the gathering problem, there is the *scattering* problem: to solve this problem, the robots, initially gathered, have to be located, in finite time, at distinct points of the continuous space [70, 108]. Lots of articles in the literature require the robots to be initially at distinct positions; therefore the scattering problem is an important problem.

Some articles deal with the *pattern formation* problem: starting from arbitrary initial positions, the robots must, in finite time, arrange themselves in the continuous space to form the pattern given in input [72, 56, 91]. Note that the robots are allowed to form a pattern that corresponds to a homothety and/or to a rotation of the pattern given in input. Sometimes the gathering is viewed as a pattern formation problem since the robots have to form a point as pattern. The pattern formation problem is an important problem since the robots are able to position themselves in the plane relatively to each other; therefore, they are able to attribute

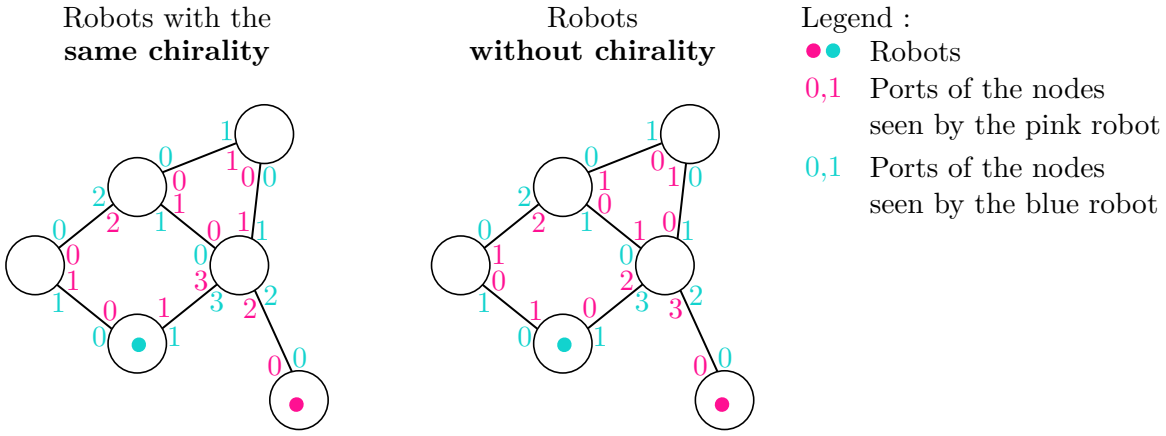


Figure 3.2: Illustration of the chirality in graphs.

themselves some roles to follow for a future task. Like for the gathering problem, some solutions to the pattern formation problem only converge toward the pattern given in input. For instance, some solutions converge through a uniform circle: the robots converge to a position such that all the robots are located evenly on the circumference of a circle [58, 59]. Such solutions cannot be composed with other algorithms to solve more complex tasks since they only converge through the pattern given in input and hence never terminate.

In the literature, there exist some articles dealing with the *leader election* problem in the plane: given a set of robots at arbitrary positions on the plane, all the robots agree, in finite time, on a specific robot which is the leader [36, 72]. Thanks to the result of Flocchini et al. [94], Dieudonné et al. [72] prove that the leader election problem is equivalent to the pattern formation problem for four and more asynchronous, oblivious, anonymous robots with the same chirality. This means that, in this setting, the pattern formation problem is solvable if and only if the leader election problem is solvable.

There also exist perpetual (i.e., that never terminate) tasks studied in the state of the art. Contrary to the algorithms that converge through a solution, the algorithms solving a perpetual problem never terminate because of the nature of the problem (that is perpetual). This is the case of the *flocking* problem in which robots have to move in the continuous space while forming a given pattern [35, 141, 36].

Discrete static environment. The gathering and rendezvous problems are also studied while considering discrete static environment. The goal of these problems in continuous space and in discrete static environment is quite similar. Indeed, in discrete static environment the gathering [135, 81] and the rendezvous [51, 69] problems are defined as follows. Robots, initially scattered, have to be, in finite time, located on a same node of the graph. When the robots are asynchronous, then the robots are also allowed to gather on edges. Similarly as for the continuous space the rendezvous problem corresponds to the gathering of two robots. Note that the approach problem (in the continuous space) can be reduced to the rendezvous problem in an infinite grid [51, 67].

In discrete static environment, the scattering problem (*a.k.a. dispersion*) is also studied [14, 133, 10] and defined hereafter. From an arbitrary initial configuration, in finite time, at most one robot must be located in each node of the graph.

The leader election problem is also studied while considering discrete static environment [15, 61] with a similar goal as in continuous space (only one robot must end up leader, and all the other robots of the system must know which robot is the leader).

While considering the discrete static environment, a fundamental problem is the exploration problem. Indeed, since the seminal work of Shannon [132], exploration of graphs by a cohort of robots has been extensively studied. There exist mainly three variants of the problem described

below. For each of these variants, the robots have to explore the graph, i.e., each node of the graph must be visited at least once by a robot. In the *exploration with stop* variant the robots have to explore, in finite time, the graph, and then they are required, also in finite time, to stop their execution once they detect that the graph has been explored [83]. In the *exploration with return* variant, the robots have to explore the graph in finite time and then they are required, also in finite time, to come back to their initial location once they detect that the graph has been explored [74]. Finally, in the *perpetual exploration* variant, each node has to be infinitely often visited by some robots [12]. It is also possible to consider the exclusive version of each variant, in which no two robots should ever be located at the same node or cross the same edge at the same time [21].

Discrete dynamic environment. While considering dynamic graphs, the gathering [110] and the rendezvous [142] problems are also studied (i.e., initially scattered, the robots have to be located in finite time on a same node of the graph).

In $\mathcal{AC}$ graphs the gathering problem is impossible [122]. Therefore, some articles study the near-gathering [122, 128], where robots must end up located on two adjacent nodes. More precisely, one article considers the near-gathering with termination (i.e., the robots must terminate their execution in finite and bounded time on two adjacent nodes)[122], while another article considers the near-gathering without termination (i.e., the robots must end up on two adjacent nodes without necessarily terminating their execution) [128].

The scattering problem is also studied [3]. Similarly as in discrete static environment, to solve the scattering problem, there must be, in finite time, at most one robot in each node of the dynamic graph.

There exist articles in the literature dealing with the exploration with stop problem [105, 90, 125, 99] where the goal is for the robots to explore the graph in finite time, and then they are required, also in finite time, to stop their execution once they detect that the graph has been explored.

Refer to Section 5.2 and Section 8.2 for more details about existing work studying robots in dynamic graphs.

3.2 Model

We first present, in Section 3.2.1, the model of robots (set of assumptions made on robots including their computational model) we consider all along this thesis. This model is an extension of the classical model of robot networks in static graphs introduced by Klasing et al. [113] to the context of dynamic graphs. Section 3.2.2 defines formally what an execution of an algorithm using robots evolving in dynamic graphs is. Then we present, in Section 3.2.3, a hierarchy of models of robots. Finally, Section 3.2.4 describes different formations of robots.

3.2.1 Assumptions

In this section, we describe the assumptions we made on robots. First we present the common assumptions that we made on robots in each of the chapters of this thesis, then we present assumptions that may differ depending on the chapter considered.

Common assumptions made on robots all along the chapters of this thesis. We consider distributed systems made of $\mathcal{R}$ autonomous mobile entities, called robots, moving in a discrete and dynamic environment modeled by an evolving graph $\mathcal{G} = ((V, E_0), (V, E_1)\ldots)$. V is a set of nodes representing the set of locations where robots may be. E_i is the set of bidirectional edges representing connections through which robots may move from one location to

another one at time i. We consider fully-synchronous robots (refer to the definition of FSYNC in Section 3.1.1) that execute the same algorithm (an algorithm being a finite sequence of ordered instructions). While executing their algorithm each robot may cross at most one edge at each instant time (i.e., at each L-C-M cycle). The robots we study possess persistent memory; hence they are able to store local variables that are not reset from one L-C-M cycle to another one. When necessary, we index the variables with the name of the robot to clarify to which robot the variable belongs to. Similarly, the robots have access to functions and predicates (that we present below), and when necessary, we denote by $name_{r,t}$ the value of a variable or a function/predicate named *name* of a given robot r after the Look phase of a given round t. The *state* of a robot at time t corresponds to the values of its variables at time t. All the robots have local vision, i.e., they view their own node, and the adjacent edges of their current node.

Other assumptions made on robots. Below we present all the other assumptions made on robots. Depending on the chapter considered these assumptions may differ, therefore, we will precise which assumptions among the one below are considered at the beginning of each chapter.

Identifications: The robots may be identified. In this case, each robot has a distinct identifier (denoted *id*) in a finite set ID of positive integers strictly greater than zero. Initially, a robot only knows the value of its own identifier. Conversely, the robots may be anonymous, i.e., they are indistinguishable from each other.

Knowledge about the environment and robots: The robots may have no prior knowledge about the graph they evolved in (size, diameter, dynamics, *etc.*) nor on the robots (number, bound on size of identifiers when the robots are identified, *etc.*). Conversely, they may have access to two information: they may know the size n of the graph in which they are evolving in, and they may be aware of the total number $\mathcal{R}$ of the robots evolving in the graph.

Topology of the dynamic graph in which the robots evolve: The robots may evolve in dynamic ring-shaped graphs. In this case, we assume that each robot has a variable *dir* that stores a direction (either *left* or *right*). For some of the algorithms this variable may also take the value $\perp$, meaning no direction. We define by $\overline{dir}$ the opposite direction of *dir*, meaning that if *dir* is equal to *right* (resp. $left$) then $\overline{dir}$ is equal to $left$ (resp. *right*). To simplify the presentation of the algorithms, we assume that the robots have access to the following local functions:

- $ExistsEdge(dir, round)$, with $dir \in \{right, left\}$ and $round \in \{current, previous\}$ which indicates if there exists an adjacent edge to the location of the robot at time t and $t-1$ respectively in the direction *dir* in G_t and in G_{t-1} respectively.
- $ExistsAdjacentEdge()$ returns true if an edge adjacent to the current node of the robot is present, false otherwise. Note that this function is used only to simplify the writing of the algorithms since $ExistsAdjacentEdge() \equiv ExistsEdge(right, current) \vee ExistsEdge(left, current)$.

We say that a robot is edge-activated during a round t if its predicate $ExistsAdjacentEdge()$ is true at time t.

Conversely, it is possible for the robots to evolve in dynamic arbitrary-shaped graphs.

Multiplicity detection: The robots may be endowed with weak local multiplicity detection, meaning that the robots are able to detect if they are alone on their current node or not, but they cannot know the exact number of co-located robots. When robots are endowed with weak local multiplicity detection, they have access to the value of the local function defined hereafter:

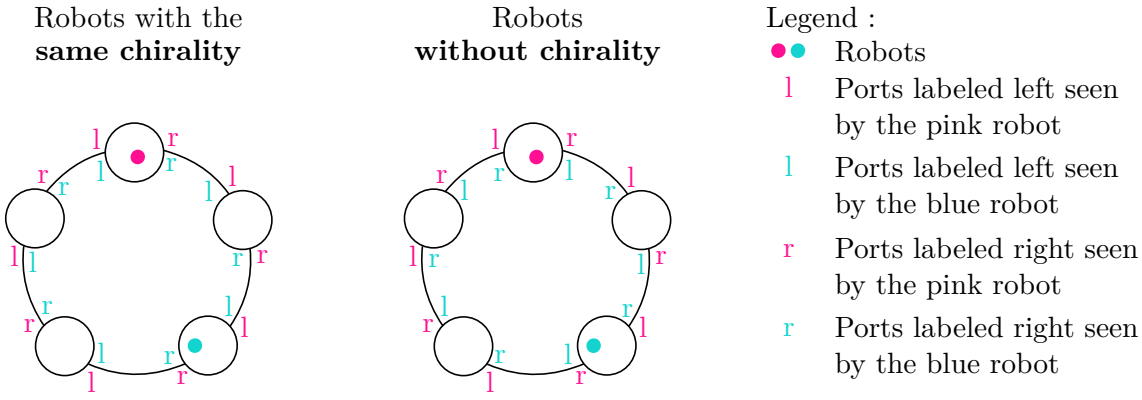


Figure 3.3: Illustration of the chirality.

- $ExistsOtherRobotsOnNode()$ returns true if there is strictly more than one robot on the current node of the robot, false otherwise.

Robots may be endowed with strong local multiplicity detection, meaning that they are able to count the exact number of robots that are located on their current node at any time t. When robots are endowed with strong local multiplicity detection, they have access to the value of the following local function:

- $NumberOfRobotsOnNode()$ that returns the exact number of robots present at the node of the robot.

Orientation: We assume that when the graph considered is a ring, each robot is able to locally label the two ports of its current node with $left$ and $right$ consistently over the ring and time. Two different robots may not necessarily agree on this labeling. In this case we say that the robots do not have the same chirality, otherwise we say that the robots have the same chirality (i.e., the robots agree on the labeling of the ports). Refer to Figure 3.3 for an illustration of what the chirality is. When robots have the same chirality and evolve in a dynamic ring, we call $Seg(u, v)$ the set of nodes (of the footprint of the dynamic ring) between node u not included and v not included considering the right direction of the robots of the system. Refer to Figure 3.4 for two examples of $Seg(u, v)$ in two different rings. At any time, we say that a robot points to the *left* (resp. *right*) if its variable *dir* is equal to this (local) direction. We say that a robot considers the clockwise (resp. counter-clockwise) direction if the (local) direction pointed to by this robot corresponds to the (global) direction seen by an external observer. Through misuse of language, whether the robots evolve in a dynamic ring or an arbitrary dynamic graph, we say that a robot points to an edge when this edge is connected to the current node of the robot by the port labeled with its current direction.

Communication: Robots may be unable to directly communicate with each other by any means. Conversely, they may be able to communicate (by direct reading) the values of their variables to each other only when they are located on a same node of the graph. In this case, the robots have access to the values of the two local functions given below:

- $NodeMate()$ which gives to the robot r the set of all the states of the robots co-located with r (the state of r is not included in this set).
- $NodeMateIds()$ which gives to the robot r the set of all the identifiers of the robots co-located with r (the identifier of r is not included in this set).

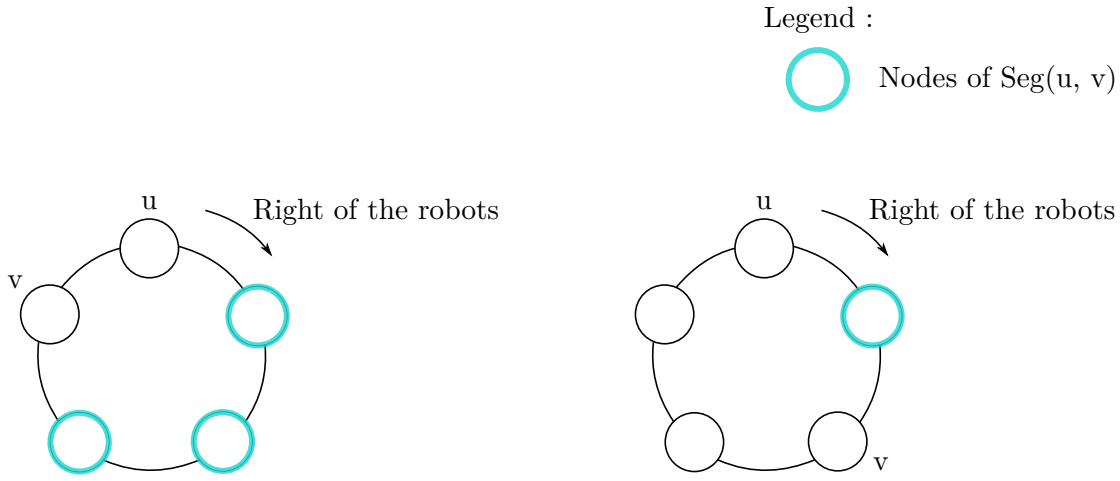


Figure 3.4: Illustration of $Seg(u, v)$ in two different rings.

3.2.2 Execution of an Algorithm

In this section we present the notions permitting to define an execution of an algorithm using robots evolving in dynamic graphs. Then we define formally this concept.

The *configuration* γ_t of the system at time t gathers the snapshot at time t of the evolving graph, the positions (i.e., the nodes where the robots are currently located), and the state of each robot at time t. We call *view* of a robot r at a time t the union of its state at time t, the values of the local functions defined in its model, and the value of the following local predicate:

- $HasMoved()$ which indicates whether r has moved between time $t-1$ and t (see explanations of the Move phase).

The algorithms, presented in this thesis, are written under the form of an ordered set of guarded rules $(label) :: guard \longrightarrow action$ where *label* is a name to refer to the rule in the text, *guard* is a predicate on the view of the robot, and *action* is a sequence of instructions modifying its state. Whenever a robot has at least one rule whose guard is true at time t, we say that this robot is *enabled* at time t. During the Compute phase, each robot enabled by the algorithm executes the *action* associated to the first rule of the algorithm whose *guard* is true in its view.

Given an evolving graph $\mathcal{G} = (G_0, G_1, \ldots)$ and an initial configuration γ_0, the *execution* ε in $\mathcal{G}$ starting from γ_0 of an algorithm is the maximal sequence $(\gamma_0, \gamma_1)(\gamma_1, \gamma_2)(\gamma_2, \gamma_3)\ldots$ (i.e., is either an infinite sequence or a sequence such that its last configuration is a terminal configuration which means a configuration with no enabled robot) in which at least one robot is enabled in each configuration (except in the last one if the sequence is finite), and where, for any $i \geq 0$, the configuration γ_{i+1} is the result of the execution of a fully-synchronous round by all enabled robots from γ_i that is composed of three atomic and synchronous phases: Look, Compute, Move defined as follows.

During the Look phase, each robot captures its view at time t.

During the Compute phase, each robot executes its algorithm that may modify some of its variables (in particular *dir*) depending on its current state and on the values of the local functions/predicates updated during the Look phase.

The Move phase consists of moving a robot in the direction it points to if there exists an adjacent edge in that direction to its current node, otherwise (i.e., the adjacent edge is missing) the robot is *stuck* and hence remains on its current node. In the case where the direction *dir* of a robot is $\perp$, the robot remains on its current node. Note that the i^{th} round is entirely executed on G_i and that the transition from G_i to G_{i+1} occurs only at the end of this round.

Note that the initial configuration γ_0 from which the execution starts belongs to a set Γ which includes all the initial configurations from which the algorithm may start.

3.2.3 Hierarchy of Models

We have presented, in Chapter 2, a hierarchy of dynamic graphs (refer to Figure 2.2 and Figure 2.5). In this section, we present a hierarchy of models of robots. A model of robots includes the computational model of robots (refer to Section 3.1.1 that presents three computational models of robots: FSYNC, SSYNC and ASYNC) and a set of capacities that robots can possess (refer to Section 3.1.3 that describes the main capacities possessed by robots in the state of the art). It is important to define a hierarchy of models of robots in order to be able to compare multiple algorithms written in different models: this hierarchy of models has deep consequences on calculability, refer to Section 4.1 for more details.

Formally, the computational model of robots corresponds to the subset of executions among all the possible executions that satisfy the synchronization of the phases of the L-C-M cycles of the robots considered. A computational model x is stronger than (denoted $\succ$) a computational model y if the set of allowed executions induced by x includes the set of allowed executions induced by y. In the case of the three computational models that we have presented in the state of the art (see Section 3.2), we have: the ASYNC model is stronger than the SSYNC and the FSYNC models, and the SSYNC model is stronger than the FSYNC model. To sum up, we have: FSYNC $\prec$ SSYNC $\prec$ ASYNC.

Similarly, each of the capacities presented in Section 3.1.3 corresponds to the subset of executions among all the possible executions that satisfy the properties induced by these capacities. In the same way, some capacities are stronger than others: a capacity x is stronger than (denoted $\succ$) a capacity y, if the set of allowed executions induced by x includes the set of allowed executions induced by y. For instance, the anonymous assumption on robots is stronger than the identified assumption on robots. Indeed, if an algorithm solves a problem thanks to anonymous robots, it obviously solves the same problem in the same model except that robots are identified.

Formally, a model of robots is the intersection of the computational model of the robots and of all the capacities possessed by the robots. A model $\mathcal{M}$ is stronger than (denoted $\rhd$) a model $\mathcal{M}'$ if it is included in $\mathcal{M}$. As an example, the model composed of asynchronous, anonymous, oblivious robots without chirality is stronger than the model composed of fully-synchronous, identified robots with persistent memory and with chirality.

Note that the relation between the models is a partial order. Indeed, some models are not comparable: for instance, the model composed of asynchronous and identified robots is not comparable to the one constituted of fully-synchronous and anonymous robots. Refer to Figure 3.5 for an illustration of a hierarchy of models of robots. In this figure, the models of robots $\mathcal{M}_{gathering}$, $\mathcal{M}_{exploration}$, and $\mathcal{M}_{self-stabilizing\ exploration}$ are models that we use in this thesis respectively in Chapter 6, in Chapter 9, and in Chapter 10.

3.2.4 Towers

In this section, we present some formations of the robots that we used to prove the correctness of our algorithms.

When a robot is alone on its current node, we say that it is isolated. At the opposite, if there are multiple robots on a same node, we say that they form a tower. Intuitively, a tower captures the simultaneous presence of all robots of a given set on a node at each time of a given interval. We require either the set of robots or the time interval of each tower to be maximal. Note that the tower is not required to be on the same node at each time of the interval (robots of the tower may move together without leaving the tower).

Definition 3.1 (Tower)**.** *A tower T is a couple (S, θ), where S is a set of robots ($|S| > 1$) and $\theta = [t_s, t_e]$ is an interval of $\mathbb{N}$, such that all the robots of S are located at the same node at each*

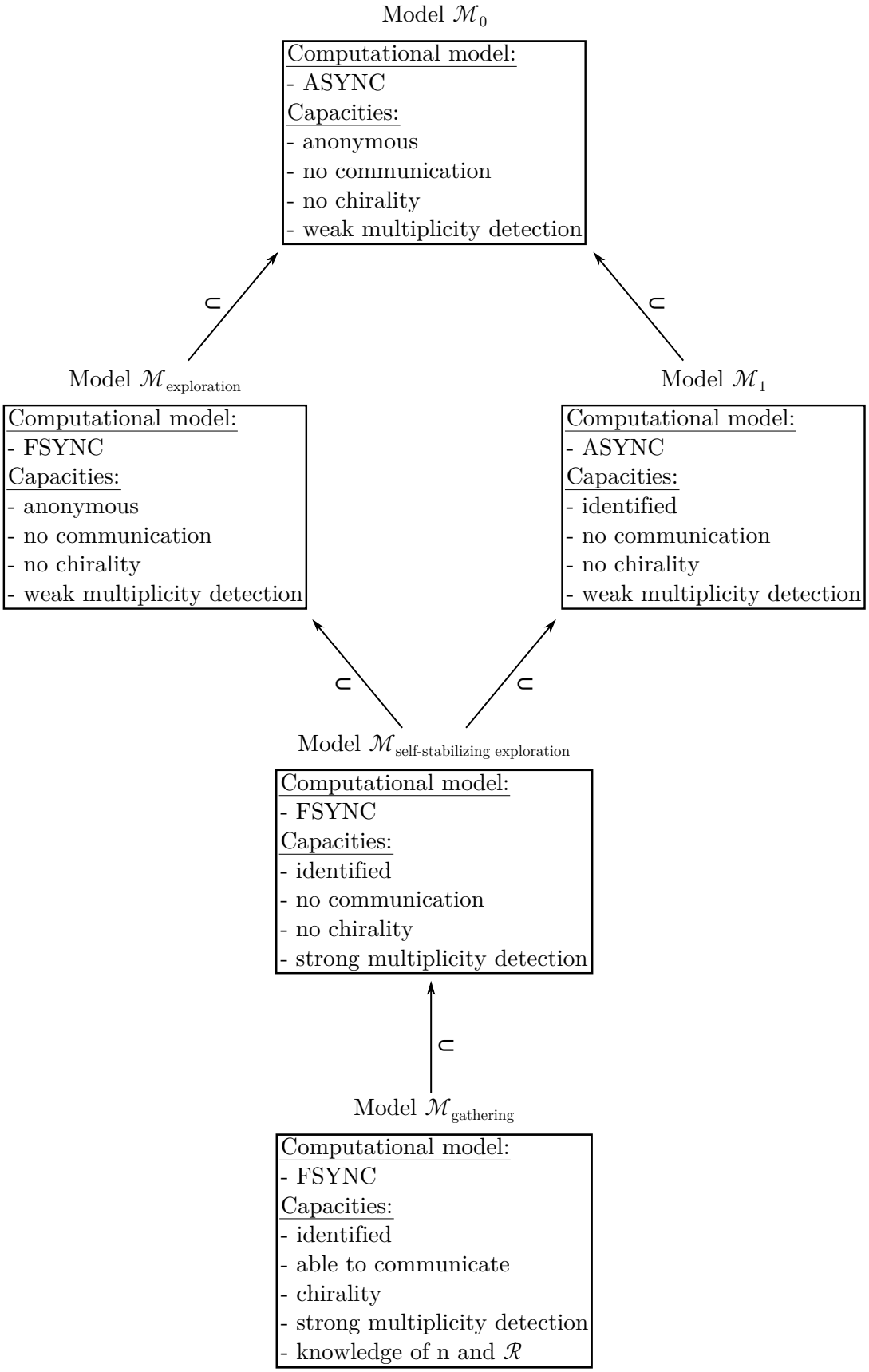


Figure 3.5: A hierarchy of models of robots.

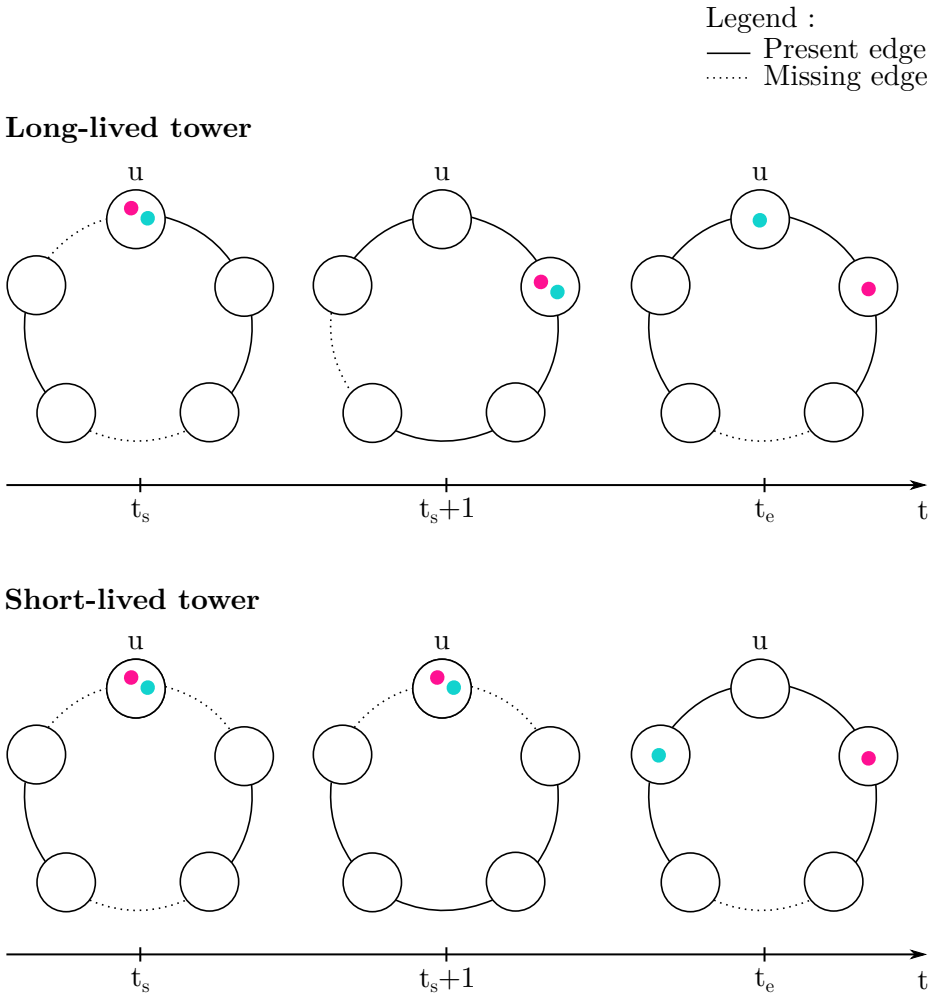


Figure 3.6: Example of a long-lived tower and a short-lived tower.

instant of time t in θ and S or θ are maximal for this property. Moreover, if the robots of S move during a round $t \in [t_s, t_e[$, they are required to traverse the same edge. We say that the robots of S form the tower at time t_s and that they are involved in the tower between time t_s and t_e.

We distinguish two kinds of towers according to the agreement of their robots on the global direction to consider at each time there exists an adjacent edge to their current location (excluding the last one). If they agreed, the robots form a long-lived tower while they form a short-lived tower in the contrary case. This implies that a short-lived tower is broken as soon as the robots forming the tower are edge-activated, while the robots of a long-lived tower move together at each edge-activation of the tower (excluding the last one). Refer to Figure 3.6 for an example of a long-lived tower and a short-lived tower.

Definition 3.2 (Long-lived tower)**.** *A long-lived tower $T = (S, [t_s, t_e])$ is a tower such that there is at least one edge-activation of all robots of S in the time interval $[t_s, t_e[$.*

Definition 3.3 (Short-lived tower)**.** *A short-lived tower T is a tower that is not a long-lived tower.*

For $k > 1$, a long-lived (resp., a short-lived) tower $T = (S, \theta)$ with $|S| = k$ is called a k-long-lived (resp., a k-short-lived) tower.

Chapter 4

Satisfiability and Impossibility Results

Contents

The goal of this thesis is to extend the notions of speculation and gracefully degrading to robot networks. Speculative algorithms are algorithms that solve a problem in any execution but are optimized (for instance in terms of the time complexity) for the most frequent executions. Gracefully degrading algorithms are conceived to circumvent impossibility results. When a problem is impossible to solve in some executions, a gracefully degrading algorithm may be conceived to solve this problem: it will solve the problem in the executions in which it is possible to do so, and in the executions in which the problem is impossible to solve, the algorithm will provide an approached solution to the problem.

To define formally speculative algorithms and gracefully degrading algorithms it is then crucial to define the notions of impossibility results and the meaning of "an algorithm solves a problem." In this chapter, we define in Section 4.1 these notions in the context of robot networks evolving in dynamic graphs. Then, in Section 4.2, we present a framework that helps to prove impossibility results in dynamic graphs.

4.1 Implications of Models and Classes of Dynamic Graphs' Hierarchies

Thanks to the hierarchy of models of robots and the hierarchy of dynamic graphs, we are able to compare multiple solutions solving a same problem. When studying distributed systems, the classical approach is to determine the necessary and sufficient conditions to solve a problem. In this thesis, we adopt the same approach but applied to robot networks evolving in dynamic graphs. All the definitions and properties presented in this paragraph are crucial since they permit to determine the strongest combination of models of robots and of classes of dynamic graphs in which a problem can be solved. Note that, since some models of robots are not comparable and some classes of dynamic graphs are not comparable, it is possible to have multiple such combinations.

We present below definitions permitting to determine the strongest combinations of models of robots and of classes of dynamic graphs in which a problem can be solved.

Definition 4.1 (Specification of a problem)**.** *The specification of a problem $\mathcal{P}$, denoted $\mathcal{SE}_{\mathcal{P}}$, is the set of all the executions that satisfy $\mathcal{P}$.*

There exists an infinity of specifications, this is why this notion is defined thanks to a set of executions induced by some properties.

Note that, some problems are comparable (i.e., some problems are stronger than others): a problem $\mathcal{P}$ is stronger than a problem $\mathcal{P}'$ if $\mathcal{SE}_{\mathcal{P}} \subset \mathcal{SE}_{\mathcal{P}'}$. For instance, the gathering problem in dynamic graphs is stronger than the near-gathering problem without termination in dynamic

graphs. Indeed, in an execution where the gathering problem is satisfied, all the robots terminate their execution on a same node of the graph in finite time. This execution also satisfies the near-gathering problem without termination. However, in an execution where the near-gathering problem without termination is satisfied, all the robots must end up on two adjacent nodes of the graph without necessarily terminating their execution. This execution does not satisfy the gathering problem.

Before presenting the fundamental notions of this section (i.e., which are the necessary and sufficient conditions to solve a problem), we need to introduce the meaning of "an execution of an algorithm under a model of robots." It corresponds to the intersection of the possible executions of the algorithms with the possible executions induced by the models of robots.

Definition 4.2 (Algorithm satisfying a problem in a class of evolving graphs under a model of robots)**.** *An algorithm $\mathcal{A}$, starting from a set of initial configurations Γ, satisfies the problem $\mathcal{P}$ (specified by $\mathcal{SE}_{\mathcal{P}}$) in a class of evolving graphs $\mathcal{C}$ under a model of robots $\mathcal{M}$, if and only if any execution of $\mathcal{A}$, starting from any initial configuration $\gamma_0 \in \Gamma$, under $\mathcal{M}$ in any evolving graph $\mathcal{G} \in \mathcal{C}$ belongs to $\mathcal{SE}_{\mathcal{P}}$.*

Definition 4.3 (Problem impossible in a class of evolving graphs under a model of robots)**.** *A problem $\mathcal{P}$ is impossible to solve in a class of evolving graphs $\mathcal{C}$ under a model $\mathcal{M}$ of robots if for any algorithm $\mathcal{A}$, starting from a set of initial configurations Γ, there exists an execution of $\mathcal{A}$, starting from an initial configuration $\gamma_0 \in \Gamma$, under $\mathcal{M}$ in an evolving graph $\mathcal{G} \in \mathcal{C}$ that does not belong to $\mathcal{SE}_{\mathcal{P}}$.*

As indicated in Section 2.1.2 and as shown on Figure 2.2, there exists a hierarchy of classes of dynamic graphs. This hierarchy permits to deduce some important properties:

Corollary 4.1. *If a problem $\mathcal{P}$ is impossible in a class of dynamic graphs $\mathcal{C}$ under a model $\mathcal{M}$ of robots, then $\mathcal{P}$ is impossible under $\mathcal{M}$ in any class of dynamic graphs $\mathcal{C}'$ such that $\mathcal{C} \subset \mathcal{C}'$.*

Corollary 4.2. *If an algorithm $\mathcal{A}$ satisfies a problem $\mathcal{P}$ in a class of dynamic graphs $\mathcal{C}$ under a model $\mathcal{M}$ of robots, then $\mathcal{A}$ satisfies $\mathcal{P}$ under $\mathcal{M}$ in any class of dynamic graphs $\mathcal{C}'$ such that $\mathcal{C}' \subset \mathcal{C}$.*

Similarly, as indicated at the beginning of this section, there exists a hierarchy in the models of robots. This hierarchy permits to deduce the following properties:

Corollary 4.3. *If a problem $\mathcal{P}$ is impossible in a class of dynamic graphs $\mathcal{C}$ under a model of robots $\mathcal{M}$, then $\mathcal{P}$ is impossible in $\mathcal{C}$ in any model of robots $\mathcal{M}'$ such that $\mathcal{M} \lhd \mathcal{M}'$.*

Corollary 4.4. *If an algorithm $\mathcal{A}$ satisfies a problem $\mathcal{P}$ in a class of dynamic graphs $\mathcal{C}$ under a model of robots $\mathcal{M}$, then $\mathcal{A}$ satisfies $\mathcal{P}$ in $\mathcal{C}$ in any model of robots $\mathcal{M}'$ such that $\mathcal{M}' \lhd \mathcal{M}$.*

4.2 A General Framework for Impossibilities in Dynamic Graphs

In static graphs there are problems that are impossible. For instance, the consensus problem (in which processes have to decide, in finite time, the same value among a set of values proposed by each of the processes) is impossible in asynchronous systems even when at most one process may crash [82]. When considering dynamic systems, obviously, the number of impossibility results increases due to the dynamics.

To prove that a problem is impossible in dynamic graphs, we have to prove that, for each algorithm $\mathcal{A}$, it exists a dynamic graph where $\mathcal{A}$ does not succeed to satisfy the specification of the problem. We can construct this dynamic graph by recurrence: construct a dynamic graph $\mathcal{G}_i$ until a time t_i, prove that until time t_i the specification of the problem is not (always) satisfied in $\mathcal{G}_i$, and then construct another dynamic graph $\mathcal{G}_{i+1}$ until a time t_{i+1} (with $t_{i+1} > t_i$) such that

$\mathcal{G}_{i+1}$ is identical to $\mathcal{G}_i$ until time t_i and prove that between time t_i and t_{i+1} the specification of the problem is still not (always) satisfied, ... Thanks to the recurrence, we are able to build a sequence of dynamic graphs in which the execution of any algorithm does not satisfy the specification of the problem on ever-growing bounded prefixes. To end the impossibility proof, we want to reach the limit, with the intuition that the execution of the algorithm on the limit of the sequence shares a common growing prefix with the executions in every graph of the sequence (and hence violates infinitely often the specification of the problem).

Braud-Santoni et al. [34] prove that this intuition is true. Indeed, they propose a generic framework to prove formally impossibilities in dynamic systems. This framework is designed for the message passing model. However, it is general enough to be used in other models. It is based on a theorem that ensures that, if we take a sequence of evolving graphs with ever-growing common prefixes (that hence converges to the evolving graph that shares all these common prefixes), then the sequence of corresponding executions of any deterministic algorithm also converges. Moreover, the execution to which it converges is the execution of this algorithm in the evolving graph to which the sequence converges.

As indicated, even if this generic framework is designed for the message passing model, it is general and may be used in other models. Indeed, the proof of the theorem of this framework only relies on the determinism of algorithms and indistinguishability of dynamic graphs. These arguments are translatable in multiple models. In particular, this framework can be used in our model (see Sections 2.2 and 3.2). Hence all along this thesis, we will use this framework in order to prove formally our impossibility results.

In our model, the theorem of Braud-Santoni et al. [34] can be formalized as follows.

First, Braud-Santoni et al. define the distance between any pair of evolving graphs: it is a function inversely proportional to the length of their longest common prefix. This distance allows to use the classical definition of a limit.

Then, given an algorithm $\mathcal{A}$, starting from a set of initial configurations Γ, under a model $\mathcal{M}$ of robots and an evolving graph $\mathcal{G}$ included in a class $\mathcal{C}$ of evolving graphs, Braud-Santoni et al. define the $(\mathcal{A}, \mathcal{M}, \mathcal{G})-$output as a function that associates to any time t the configuration γ_t of the system at time t and the views at time t of each robot resulting from the execution of $\mathcal{A}$ in $\mathcal{G}$ starting from a configuration $\gamma_0 \in \Gamma$. Let $\mathcal{O}$ be the set of all $(\mathcal{A}, \mathcal{M}, \mathcal{G})-$outputs over all the evolving graphs $\mathcal{G} \in \mathcal{C}$ and over all the initial configurations $\gamma_0 \in \Gamma$. A similar distance can be defined on $\mathcal{O}$ in a similar way as previously.

The theorem proved by Braud-Santoni et al. is then the following one.

Theorem 4.1 ([34]). *For any deterministic algorithm $\mathcal{A}$ under a model $\mathcal{M}$ of robots, if a sequence $(\mathcal{G}_n)_{n\in\mathbb{N}}$ of evolving graphs belonging to a class of dynamic graphs $\mathcal{C}$ converges to a given evolving graph $\mathcal{G}_\omega \in \mathcal{C}$, then the sequence $(o_n)_{n\in\mathbb{N}}$ of the $(\mathcal{A}, \mathcal{M}, \mathcal{G}_n)-$outputs converges to $o_\omega \in \mathcal{O}$. Moreover, o_ω is the $(\mathcal{A}, \mathcal{M}, \mathcal{G}_\omega)-$output.*

See Figure 4.1 for an illustration of the theorem of Braud-Santoni et al. to construct impossibility result.

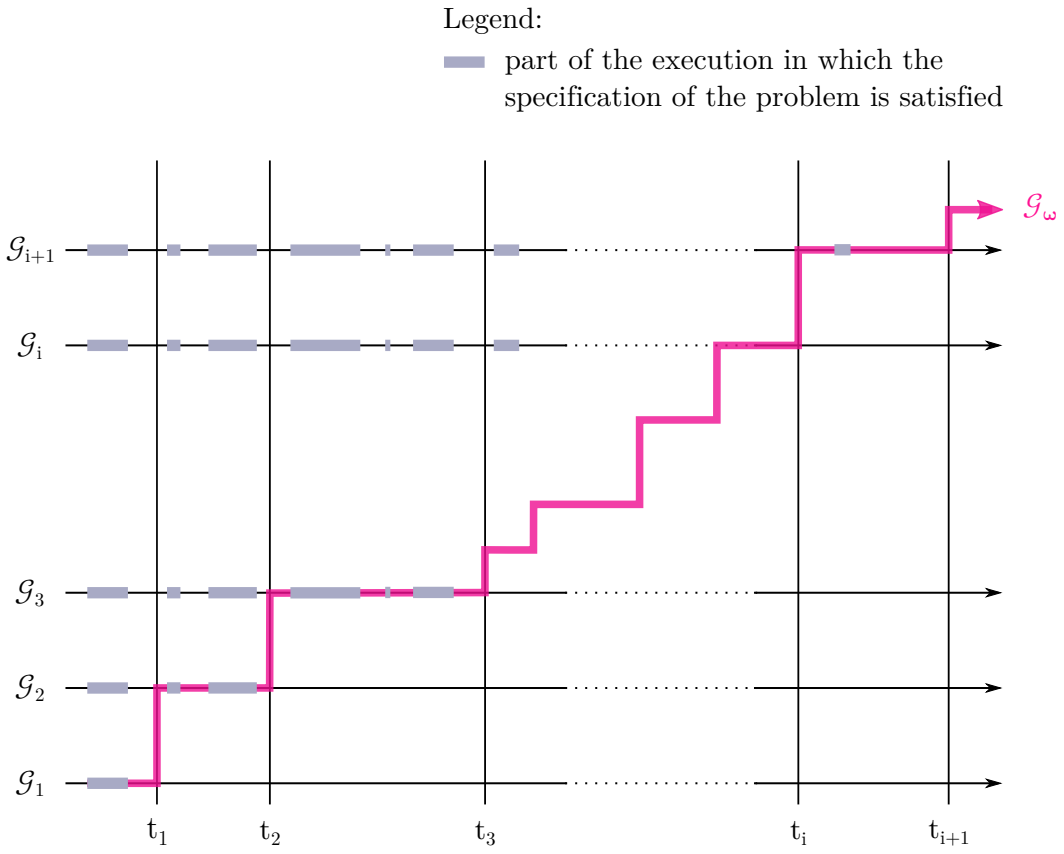


Figure 4.1: Illustration of the theorem of Braud-Santoni et al. [34].

Part II

Graceful Degradation

Chapter 5

Introduction

Contents

In this part, we present, in Chapter 6, a gracefully degrading algorithm in dynamic rings. Intuitively, a gracefully degrading algorithm must satisfy the problem studied in dynamic graphs in which it is possible to do so, and when the dynamics increase and make the problem impossible, the algorithm should provide a best effort solution (i.e., an approached solution to the original problem). The conception of a gracefully degrading algorithm for a given problem is motivated by the impossibility of this problem in a certain class of dynamic graphs.

Our gracefully degrading algorithm solves the gathering problem which consists for the robots of the system to be located on a same node of the graph in finite time. The conception of a gracefully degrading algorithm implies to consider approached solution(s) to the original problem. We only consider approached solutions to the gathering problem that weaken the termination of this problem, i.e., we authorize at most one robot to never terminate its execution, and all the robots that terminate their execution do so on a same node of the graph. Hence, we keep unchanged the essence of the problem (robots terminate on a same node). The conception of this algorithm is motivated by the impossibility of the gathering problem in $\mathcal{AC}$ rings [122].

In this chapter, we first present, in Section 5.1, the state of the art about gracefully degrading algorithms, then, in Section 5.2, we present the state of the art about gathering in dynamic graphs. Finally, we present, in Section 5.3, our motivation to conceive a gracefully degrading algorithm in dynamic graphs.

5.1 Graceful Degradation

5.1.1 Definition

As indicated in Section 2.1.2, there are multiple classes of dynamic graphs, each of them regroups dynamic graphs having the same temporal connectivity properties. The weakest class of dynamic graphs is the class of $\mathcal{ST}$ graphs: in this class the connectivity assumptions are strong since each edge of a $\mathcal{ST}$ graph is always present. Obviously, when the connectivity assumptions become weaker (i.e., when the edges are less present in the graphs), some problems, solvable in $\mathcal{ST}$ graphs, become impossible. For instance, the gathering problem (where robots must be located in finite time on a same node of the graph) is impossible to solve in $\mathcal{AC}$ rings [122]. Call $\mathcal{P}$ a problem that becomes impossible when the dynamics increase.

Sometimes, applications are executed in environments in which the dynamics cannot be known in advance, or in environments in which the dynamics evolve with times. As an example, if we consider the dynamic graph modeling a map in which streets are opened and closed

over time due to traffic jam in a town, at some hours there is little traffic and the map can be represented thanks to a $\mathcal{ST}$ graph, but at other moments the traffic is so intense that the map is modeled by a $\mathcal{COT}$ graph. Assume that the problem $\mathcal{P}$ may be considered in a changing environment such that the dynamics of this environment correspond sometimes to a dynamic in which $\mathcal{P}$ is solvable but at some other time it corresponds to a dynamic that makes the problem $\mathcal{P}$ impossible.

In order to be able to consider the problem $\mathcal{P}$ in such dynamic environment, it would be convenient to have an algorithm that solves $\mathcal{P}$ in some dynamic environments in which it is solvable, and provides a best effort solution to $\mathcal{P}$ when the dynamics increase. In other words, it would be convenient to have an algorithm $\mathcal{A}$ that simultaneously satisfies the specification of $\mathcal{P}$ in some dynamic environments in which it is solvable, and satisfies the specification of a weaker problem than $\mathcal{P}$ but that is related to the problem $\mathcal{P}$ when the dynamics increase. Such algorithms are called gracefully degrading algorithms, and have been introduced by Biely et al. [20].

Note that $\mathcal{A}$ may satisfy different weaker specifications of $\mathcal{P}$ in different dynamic classes of dynamic graphs (hopefully with stronger requirements when temporal connectivity increases).

More formally, we can define a gracefully degrading algorithm as follows:

Definition 5.1 (Gracefully degrading algorithm)**.** *Given a model $\mathcal{M}$, a problem $\mathcal{P}_0$, a set $\mathcal{S}_\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$ of problems either equal to or weaker than $\mathcal{P}_0$, with $|\mathcal{S}_\mathcal{P}| \geq 1$, and a set $\mathcal{S}_\mathcal{C} = \{\mathcal{C}_0, \ldots, \mathcal{C}_k\}$ of classes of dynamic graphs such that:*

1. *$|\mathcal{S}_\mathcal{C}| \geq 2$.*
2. *For all i, with $1 \leq i \leq k, \mathcal{C}_0 \subset \mathcal{C}_i$.*
3. *For all i, with $0 \leq i \leq k$, and for all j, with $0 \leq j \leq k$, if $\mathcal{C}_i \subset \mathcal{C}_j$ then $\mathcal{P}_i$ is either stronger than or equal to the problem $\mathcal{P}_j$ (i.e., $\mathcal{SE}_{\mathcal{P}_i} \subseteq \mathcal{SE}_{\mathcal{P}_j}$).*
4. *There exists at least one i, with $1 \leq i \leq k$, such that $\mathcal{P}_0$ is impossible to solve in $\mathcal{C}_i$ under $\mathcal{M}$.*

a $(\mathcal{P}_0, \mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{C})$-gracefully degrading algorithm $\mathcal{A}$ for $\mathcal{P}_0$ under a model $\mathcal{M}$ is an algorithm such that:

- *For all i, with $0 \leq i \leq k$, $\mathcal{A}$ satisfies $\mathcal{P}_i$ under $\mathcal{M}$ in $\mathcal{C}_i$.*

We say that an algorithm is gracefully degrading if there exist a problem $\mathcal{P}_0$, a set of problems $\mathcal{S}_\mathcal{P}$, and a set of classes of dynamic graphs $\mathcal{S}_\mathcal{C}$ such that this algorithm is $(\mathcal{P}_0, \mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{C})$-gracefully degrading. When necessary, we clarify the classes of dynamic graphs saying that an algorithm is gracefully degrading with respect to a family of classes of dynamic graphs $\mathcal{C}_i, \ldots, \mathcal{C}_k$ if there exist a problem $\mathcal{P}_0$, and a set of problems $\mathcal{S}_\mathcal{P}$ such that this algorithm is $(\mathcal{P}_0, \mathcal{S}_\mathcal{P}, \{\mathcal{C}_i, \ldots \mathcal{C}_k\})$-gracefully degrading.

Above, we give a formal definition of a gracefully degrading algorithm for robot networks. This definition is suitable to the intuition given of this notion. Indeed, the points 1 and 2 of the definition permit to indicate that the problem must be studied in a dynamic environment that may change with time, and such that there is a relation between the changes (considering a class of dynamic graph studied, there is at least another class such that the two classes are comparable). Moreover, the combination of points 3 and 4 imposes that a set of weaker versions of the original problem are considered. A gracefully degrading algorithm for this problem should solve the original problem in the weakest dynamics studied and each of the weaker versions of the problem when the dynamics of the environment increase: more the dynamics increase more the problem solved is weak. Besides, the conception of a gracefully degrading algorithm is motivated by an impossibility result, and the point 4 of the definition imposes that the original problem

is impossible to solve in at least one of the classes of dynamic graphs considered. This point is mandatory to avoid the conception of gracefully degrading algorithms that consider only classes of dynamic graphs in which the problem studied is solvable and that are gracefully degrading only because they cannot succeed to find an appropriate solution to the problem.

5.1.2 State of the Art

The gracefully degrading approach has been introduced by Biely et al. [20]. They prove that the consensus problem (where processes have to decide irrevocably, in finite time, the same value among a set of values initially proposed by each of the processes) is not solvable in synchronous directed dynamic systems when the connectivity assumptions between the communicating processes are weak. Then they provide a consensus algorithm that gracefully degrades to k-set agreement (where processes have to decide irrevocably, in finite time, k values among a set of values initially proposed by each of the processes) when the connectivity between the communicating processes decreases. Note that the value of k is not fixed, and changes from execution to execution depending on the dynamics of the system.

Biely et al. [20] have introduced the graceful degradation in the context of dynamic systems. However, the idea of gracefully degrading the specification of a problem depending on the environment is not a new idea. Indeed, in indulgent algorithms [5, 118, 131] the same idea is exploited with respect to the asynchronism of the system.

More precisely, indulgent algorithms have been defined by Alistarh et al. [5] as algorithms that guarantee correctness when the system is synchronous, and when the system is asynchronous and the problems solved by the algorithms become impossible then they maintain the safety of the problems.

Alistarh et al. [5] present a technique to transform a synchronous algorithm into an indulgent algorithm. Considering a system where up to $t = N/2 - 1$ among N processes may crash (i.e., may stop to perform their algorithm), they apply their transformation on two well-known problems: the renaming problem (where each correct process that starts its execution with a unique identifier i, with $0 \leq i \leq I$ for some large I, has to end up with a unique new identifier j such that $0 \leq j \leq J$, where $J < I$ and J is as small as possible), and the consensus problem (where correct processes must decide irrevocably, in finite time, one value among a set of values initially proposed by each of the processes). In case of a synchronous system, their indulgent renaming algorithm satisfies the specification of the renaming problem given identifiers between 1 and N. In case of an asynchronous system where processes may be subject to crash, renaming the correct processes with identifiers between 1 and N is not possible. In this setting, the indulgent algorithm of Alistarh et al. renames the correct processes with identifiers between 1 and $N + t$. In the case where the system is synchronous, their indulgent consensus algorithm ensures the weak termination (each correct process will eventually decide but it may continue to run the protocol after the decision) and quiescence (eventually the protocol stops sending messages, but the processes may continue to participate in the protocol). In the case where the system is asynchronous and some processes may crash, the consensus problem is impossible to solve [82]. In this setting, their indulgent consensus algorithm does not ensure weak termination or quiescence anymore: the decision of the processes is delayed until the system becomes synchronous, preserving the safety of the problem.

Similarly, Lamport [118], and De Prisco et al. [131], present an indulgent algorithm solving the consensus problem when processes may crash: when the system is synchronous the consensus is solved, however, when the system is asynchronous the termination of the algorithm is not ensured.

With the same underlying idea than gracefully degrading algorithms and indulgent algorithms, Vaidya and Pradhan [140] propose an algorithm that degrades the specification of a problem when the number of Byzantine faults (i.e., arbitrary faults) in the system increases. They consider the consensus problem in synchronous systems where processes may be subject

to Byzantine faults. Lamport et al. [119] have proved that to solve the consensus problem in synchronous systems in the presence of b Byzantine faults, the system must be composed of at least $3 * b + 1$ processes. The algorithm proposed by Vaidya and Pradhan solves the consensus problem when the number of Byzantine faults is less or equal to b, however they authorize the correct processes to agree on two different values when the number of Byzantine faults in the system increases above b.

In the state of the art about robots, there is neither indulgent algorithms nor algorithms explicitly exploiting the gracefully degrading approach. However, some of the articles dealing with the gathering in dynamic graphs are gracefully degrading without knowing it (refer to Section 5.3 for an overview of these articles).

5.2 State of the Art About Gathering in Dynamic Graphs

Although there exists a huge literature on gathering/rendezvous in continuous space [43, 71, 4] and in discrete static environments [113, 53, 135], these studies fall out of the scope of this thesis. In this section, we present only articles dealing with gathering in dynamic environments.

There are few articles solving the gathering/rendezvous problem while considering dynamic environments. However, there are some articles dealing with the gathering problem in static graphs while some elements of the model introduce a kind of dynamics. These articles are interesting to study since they may provide some clues to deal with real dynamic environments. We present such articles in Section 5.2.1. In Section 5.2.2, we present the state of the art about gathering in dynamic graphs.

5.2.1 Towards Dynamic Graphs

The articles we present in this section consider the gathering/rendezvous problem in static graphs where some elements of the model permit to introduce a kind of dynamics.

Mobile malicious entity. Indeed, Das et al. [54, 53] consider the gathering problem while there exists in the undirected network one mobile malicious entity that prevents robots from visiting the node where it is located. In particular, the robots cannot move to a node in which the malicious entity is located. The malicious entity moves arbitrarily fast, but it cannot stop on a node where a robot is already located. Moreover the edges are crossed by the robots as well as by the malicious entity in a FIFO way.

Therefore, in this setting, it is possible to simulate a missing edge: a missing edge is an adjacent edge to the node where the malicious entity is located. It is also possible to simulate eventual missing edge: in the case where the malicious entity does not move after a certain time, all the adjacent edges to its position are eventual missing edges. However, this model does not permit to represent fully dynamic environment since the disappearance of edges is dependent on the positions and movements of the robots. Indeed, since the malicious entity cannot be located on the same node as a robot, missing edges are dependent on the positions of the robots. And since the edges are crossed in a FIFO way, missing edges are dependent on the movements of the robots (and hence of the algorithm). Moreover, in this setting, it is not possible to have two robots located on two adjacent nodes such that they are separated by a missing edge.

In this "dynamic setting" Das et al. [54] consider asynchronous, anonymous robots. The robots possess finite memory and are able to communicate when they are located on a same node. They do not have any prior knowledge about the graph in which they are evolving, in particular, they do not know the size of the graph. Initially there is at most one robot in each node. The robots do not know the total number of robots present in the system. In this context, the authors first give a set of initial configurations Γ (whatever the topology of the graph) for which the gathering problem is not solvable. Γ is the set of initial configurations for which the malicious entity may

disconnect the graph such that not all the robots are on the same connected component. As indicated by the authors, in ring-shaped graphs no execution of any algorithm can reach the configurations of Γ. They then provide a gathering algorithm for robots with chirality evolving in rings with one marked node. They also consider robots without chirality evolving in rings, and prove that if there are an even number of robots, the gathering problem is not solvable even if there is one marked node in the ring. They thus provide an algorithm solving the gathering problem for rings with one marked node in the case where there are an odd number of robots without chirality in the system. While executing this algorithm, the number of edges crossed by the robots is in $O(\mathcal{R} * n)$ (with $\mathcal{R}$ the total number of robots in the system and n the size of the ring). Finally, they study robots with chirality evolving in grids. For this topology, they prove that, to solve the gathering, robots need to have a visibility range at distance 2 if the robots are initially located on consecutive nodes. They provide then an algorithm solving the gathering in this setting (the graph is a grid and the robots have the same chirality, are endowed with a visibility range at distance 2, are initially located on consecutive nodes and are not initially in a configuration that belongs to Γ). This algorithm is achieved in $O(\mathcal{R}^2)$ edge traversals, whatever the size of the grid. All their results are true for a number of robots greater or equal to two.

Das et al. [53] conduct a similar study. In the same setting, except that the robots are fully-synchronous, they prove that the gathering problem is not solvable when the number of robots without chirality is even and the size of the ring is odd, even if the ring possesses one marked node. When the size of the ring is even or the number of robots without chirality is odd, they provide an algorithm solving the gathering problem when the ring possesses one marked node. All their results are true for a number of robots greater or equal to two. When the number of robots is strictly greater than two, their algorithm achieves the gathering in $O(n)$ rounds, and with $O(n * \mathcal{R})$ edge traversals.

Delay faults. Chalopin et al. [41] consider the problem of rendezvous (i.e., gathering of two robots) in undirected graphs where the robots without chirality may be subject to delay faults: if a robot is faulty at a time t then it does not move during the L-C-M cycle of time t. More precisely, they consider three kinds of delay faults: random faults (where each robot may be faulty at each round with a probability p such that $0 < p < 1$), unbounded faults (where a robot is faulty a finite number of consecutive rounds), bounded faults (where a robot is faulty a finite and bounded number of consecutive rounds). Moreover, a faulty robot is aware that it is faulty.

Since Chalopin et al. consider only two robots, and since these two robots stop their execution when they are located on a same node, their setting can simulate dynamic directed graphs. Indeed, a missing arc is an adjacent outgoing arc to a faulty robot. However, their setting cannot simulate undirected dynamic graphs since the faults they consider cannot simulate a missing edge: only one of the two robots located on two adjacent nodes and trying to cross the same edge in opposite directions during the same round may be faulty, which is not the behavior induced by a missing edge. Note that, if there are more than two robots in the system, and if when two robots (or more) meet they do not stop their execution, then their setting cannot simulate directed dynamic graphs anymore. Indeed, if two robots are located on a same node and try to cross the same arc at the same round, they should behave in the same way: either the arc is missing and both robots should be faulty to simulate this, or the arc is not missing and none of the robots should be faulty to simulate this, but in their model, it is possible to have only one of these robots faulty.

In this setting, Chalopin et al. [41] consider fully-synchronous robots that possess distinct positive identifiers. They only know their own identifier. They evolve in any anonymous graph, and do not know the topology of the graph in which they evolve or the size of the graph. Moreover, even if the robots are fully-synchronous, they do not start necessarily their execution at the same round. When the robots are subject to random faults, the authors provide an algorithm solving the rendezvous problem in a number of edge traversals polynomial in the size of the graph and

polylogarithmic in the larger identifier of the robots, with a very high probability. When robots are subject to unbounded faults, the authors prove that the rendezvous problem is unsolvable, even when the graph is a ring (as indicated by the authors, in the case where the robots have the same chirality and know the size of the graph, it is possible to solve the rendezvous problem on rings even if robots are subject to unbounded faults). However, in this setting, the authors give a rendezvous algorithm functioning in arbitrary trees. This algorithm solves the rendezvous problem in $O(n * l)$ edge traversals, where n is the size of the network and l is the smaller identifier of the robots. The authors show that the lower bound to solve the rendezvous problem in arbitrary trees is in $\Omega(l)$ edge traversals. Finally, they provide a rendezvous algorithm for any graph topology when robots are subject to bounded faults. Their algorithm executes, to solve the rendezvous problem, a number of edge traversals polynomial in n, and logarithmic in the bound of consecutive rounds during which a robot may be faulty and in the larger identifier of the robots.

5.2.2 Dynamic Rings

There exist in the literature 4 articles studying the gathering or rendezvous problem in dynamic graphs. Each of these articles considers robots evolving in dynamic rings, and deal with a different kind of dynamics. We detail below each of these articles depending on their dynamics.

With probabilistic appearance of edges. Yamauchi et al. [142] consider the rendezvous problem in undirected, simple, anonymous, and dynamic rings. In the dynamic rings they consider, at each instant time, the presence of each edge is randomly decided with a probability p.

Since the edges of a dynamic ring appear and disappear, the authors propose two different models in which the labels of the ports may change with time. The first model they consider, called the fixed port numbering, is a model in which the labels of the ports of the edges never change. In other words, the labels of the ports of the edges correspond to those in the footprint of the dynamic graph. The second model they consider, called the non-fixed port numbering, give randomly a number (not already assigned) to the ports of the edges of the dynamic graph that appear at a given time (the edges that appear at a time t and are still present at a time $t+1$ keep unchanged their labels that were assigned at time t).

The authors consider two fully-synchronous anonymous robots without chirality. Each robot knows that there are only two robots in the system. Robots are endowed with persistent memory. Robots are able to see the adjacent edges incident to their current node. They are endowed with local strong multiplicity detection (i.e., the robots know the exact number of robots located on their current node).

The authors provide an algorithm where, at each round until the rendezvous occurs, when a robot crosses an edge it remembers the port from which it arrives at the node, and then waits to cross an edge with a different port label. They analysis this algorithm under the two different models of port numbering they proposed.

While considering the fixed port numbering model, their algorithm is equivalent to the following one: each robot chooses a direction and keeps to move in that direction until the rendezvous occurs. The authors prove that if the two robots consider initially opposite global directions then the rendezvous occurs in $O(n)$ rounds; otherwise it occurs in $O(n*d)$ rounds, where d is the initial distance (in the footprint) between the two robots.

While considering the non-fixed port numbering model, the authors prove that when p is large, then the behavior of the robots is quite identical as in the fixed port numbering model. When p is small their algorithm is equivalent to a random walk. Surprisingly, when p is small, the authors prove that the rendezvous occurs in an expected time smaller than in the case of the fixed port numbering model.

In $\mathcal{RE}$ rings. Izumi et al. [110] consider the gathering problem in anonymous, and undirected $\mathcal{RE}$ rings. Each node of the ring is endowed with a whiteboard on which robots may write and read some data when they are located on this node.

The authors focus on robots with chirality, possessing persistent memory and which know the total number of robots $\mathcal{R}$ in the system. The system is asynchronous. Some additional assumptions are made on this asynchrony to coordinate the L-C-M cycles of the robots and the appearance/disappearance of edges:

- Edges cannot disappear while they are crossed by a robot.
- The delay between the transitions from one static graph to another static graph of the evolving graph is long enough to permit the robots to visit all the nodes that are part of the connected component in which they are located.

In this setting, the authors consider three different models: the global detection model in which the robots are able to detect the time at which the evolving graph transitions from one static graph to another different one; the semi-local detection model in which the robots are able to detect the time at which the edges of the connected component in which they are located change (i.e., appear or disappear); and the local detection model in which the robots are able to detect the time at which the edges of the node in which they are located change.

First, the authors show that if the positions of the robots in the initial configuration are periodic and if the connected components in each G_i (with $i \in \mathbb{N}$) of the evolving graph represent a periodic sequence of segment lengths then the gathering is impossible for some specific periods. Their result is true even if the robots are studied in the global detection model.

Then the authors present some deterministic algorithms solving the gathering in the semi-local and local detection models. In each of these algorithms, in the case where the gathering is not possible, the robots are allowed to continue their execution forever.

In $\mathcal{AC}$ rings. Di Luna et al. [122] consider robots evolving in $\mathcal{AC}$ undirected rings. The rings studied are not anonymous: the nodes where the robots are initially located are identically marked. Initially, there is at most one robot on each node of the ring. Hence, there are as many marked nodes as robots in the system.

The authors focus on fully-synchronous robots, i.e., all robots execute their L-C-M cycles synchronously and atomically. The robots they consider are anonymous, not able to directly communicate, endowed with persistent memory, and endowed with strong local multiplicity detection, i.e., each robot is able to know the exact number of robots located on its current node. More precisely, in the model considered by Di Luna et al. the nodes are divided into three parts (two parts for each port of the node and a part at the middle of the node), and each robot is able to know the exact number of robots that are located on each part of its current node. Robots are not able to know if the adjacent edges to their current node are present or missing. Moreover, in each of the algorithms presented by Di Luna et al., the robots know either $\mathcal{R}$ (the number of robots present in the system) or n (the size of the ring).

In this setting, the authors prove that the gathering problem is impossible even if the robots have the same chirality, are endowed with cross detection (i.e., ability to detect, when traversing an edge, whether some robot is traversing it in the other direction) and know both $\mathcal{R}$ and n. Intuitively, this impossibility holds because in an $\mathcal{AC}$ ring it is possible to prevent two given robots to meet themselves by retrieving at each instant time the edge (if any) that permits them to meet, since it is allowed to have one missing edge at each instant time.

To circumvent this impossibility, the authors consider a weaker specification of the gathering problem, called the near-gathering: the robots must end up on two adjacent nodes of the dynamic ring. More precisely, they consider the near-gathering problem with termination (i.e., in finite and bounded time all the robots terminate their execution on two adjacent nodes of the dynamic ring).

	No chirality	Chirality
Cross detection	$O(n)$	$O(n)$
No cross detection	$O(n^2)$	$O(n * log(n))$

Table 5.1: Number of rounds necessary for the algorithms provided by Di Luna et al. to solve the near-gathering depending on chirality and cross detection capacities of the robots.

The goal of their study is to determine the feasibility of the near-gathering depending on chirality, and cross detection.

First, the authors prove that, whatever the chirality and cross detection capacities of the robots, if the initial positions of the robots are periodic then the near-gathering is not solvable. Denote Γ the set of initial configurations in which the positions of the robots are periodic.

They also prove that, if the robots have no chirality and are not endowed with cross detection, in addition to the set of initial configurations in Γ, the near-gathering is impossible when the initial positions of the robots present a unique axis of symmetry passing through two edges of the ring.

Then for each combination of the capacities of the robots (chirality and cross detection), they provide an algorithm solving, in finite and bounded time, the near-gathering for the initial configurations in which it is possible to do so. For each combination of the capacities of the robots, we present, in Table 5.1, the number of rounds necessary for the algorithms provided by Di Luna et al. to solve the near-gathering. Moreover, when their algorithms are executed starting from an initial configuration from which the near-gathering is impossible, the robots succeed, in finite time, to detect the impossibility.

In $\mathcal{COT}$ rings. As indicated previously, Di Luna et al. [122] prove that the gathering is impossible to solve in $\mathcal{AC}$ rings. Their arguments to prove this impossibility are quite general, and can be applied in most models of robots. Moreover, since $\mathcal{COT}$ rings include $\mathcal{AC}$ rings, the gathering is also impossible to solve in $\mathcal{COT}$ rings.

Because of this impossibility, Ooshita and Datta [128] consider the near-gathering problem in $\mathcal{COT}$ anonymous, undirected rings. More precisely, they consider the near-gathering without termination (i.e., in finite time all the robots must end up on two adjacent nodes of the dynamic ring without necessarily terminating their execution).

The authors consider fully-synchronous robots without chirality. The robots possess distinct identifiers. Initially, each robot knows its own identifier, but does not know the identifier of the other robots. None of the robots know n or $\mathcal{R}$. Robots may start at arbitrary positions, in particular, it is possible to have multiple robots located initially on the same node. Robots are endowed with persistent memory. They are not able to know if the adjacent edges to their current node are present or missing. Finally, they are able to communicate the values of their variables to the robots that are located on their current node.

In this setting, three models are considered: (*i*) there are whiteboards on nodes, in which robots can read and write information; (*ii*) robots and whiteboards on nodes have specific initial states; (*iii*) robots are allowed not to terminate.

In this context, the authors address self-stabilizing algorithms: a self-stabilizing algorithm is an algorithm that tolerates transient faults (i.e., arbitrary faults such that there exists a time from which these faults do not occur anymore). Such algorithm must then tolerate arbitrary initial states of the robots and whiteboards on nodes. If an algorithm needs the robots and whiteboards on nodes to have specific initial states, then it is not self-stabilizing.

In this setting, the authors prove three impossibility results. First, they show that if the nodes have no whiteboards then the near-gathering is impossible to solve. This result is true even if robots are allowed not to terminate and if robots and whiteboards on nodes start with

specific initial states. Then, they prove that there is no self-stabilizing algorithm solving the near-gathering. This is true even if the conditions (i) and (iii) are allowed. Finally, they show that there is no algorithm solving the near-gathering where all the robots terminate. Their result is true even when the conditions (i) and (ii) hold.

At the sight of these impossibility results, the authors provide an algorithm solving the near-gathering in a model where the conditions (i), (ii) and (iii) hold.

5.3 Motivation for a Gracefully Degrading Gathering Algorithm

As indicated in Section 5.1, the motivation to design a gracefully degrading algorithm (introduced by Biely et al. [20]) is an impossibility result in some classes of dynamic graphs.

In the literature, all the algorithms in $\mathcal{ST}$ (finite) graphs for the gathering problem using fully-synchronous robots terminate in a finite and bounded amount of time [46, 2, 49, 81]. In other words, the specification of the gathering problem solved in these solutions is the following one.

Definition 5.2 (Gathering $\mathbb{G}$)**.** *All robots terminate their execution on the same node of the graph in finite and bounded time.*

Di Luna et al. [122] proved that $\mathbb{G}$ is impossible to solve in $\mathcal{AC}$ rings. Therefore, to solve this problem in environments in which the dynamics may change with time or in environments in which the dynamics are not known in advance, a gracefully degrading approach is convenient.

None of the algorithms of the state of the art solving the gathering or rendezvous problem in dynamic graphs (see Section 5.2) is explicitly gracefully degrading: each of these algorithms is analyzed only in one specific class of dynamic graphs. However, some of them are gracefully degrading without knowing it. Hence, in the following, we study deeper each of these algorithms, analyzing them in classes of dynamic graphs other than those studied by the authors. Note that we consider only weaker variants of the gathering problem that seem quite natural, and we focus only on the classes of dynamic graphs studied in this thesis (i.e., $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$).

Algorithm proposed by Yamauchi et al. [142]: Their algorithm is not gracefully degrading with respect to any combination of $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$: it solves $\mathbb{G}$ in the class of dynamic graphs they focus on (which are graphs with a probabilistic appearance of edges), but in $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$ graphs, their algorithm does not succeed to solve any non-trivial weaker version of the gathering problem. Indeed, when executing their algorithm in $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$ graphs, it is possible for the two robots (that are supposed to meet) to be always located on different nodes.

Algorithms proposed by Izumi et al. [110]: In the case where the initial configuration of the robots is periodic, their algorithms do not succeed to solve $\mathbb{G}$. In particular, this is the case in $\mathcal{ST}$ graphs. When the robots do not start from such a problematic configuration, their algorithms are gracefully degrading with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$: they solve $\mathbb{G}$ in $\mathcal{ST}$ and $\mathcal{BRE}$ rings, a weaker version of the gathering problem where all the robots have to terminate their execution in finite (but not necessarily bounded) time on the same node of the ring in $\mathcal{RE}$ rings, and a weaker version of the gathering problem where the robots must end up on at most two different nodes (not necessarily adjacent) and where only a part of the robots may terminate their execution in $\mathcal{AC}$ and $\mathcal{COT}$ rings. More precisely, in $\mathcal{AC}$ or $\mathcal{COT}$ rings, the robots choose a node where the gathering must occur. Only the robots that succeed to reach this chosen node can terminate their execution. This implies that there is no guarantee concerning the number of robots that terminate their execution in $\mathcal{AC}$ and $\mathcal{COT}$ rings: for these classes of dynamic graphs, between none and $\mathcal{R}$ robots may terminate their execution.

Algorithms proposed by Di Luna et al. [122]: In the case where the initial configuration of the robots is periodic, their algorithms do not succeed to solve $\mathbb{G}$. In particular, this is the case in $\mathcal{ST}$ graphs. However, when the initial configuration of the robots is not periodic, their algorithms (except the one considering robots endowed with cross detection and without chirality) are gracefully degrading with respect to $\mathcal{ST}$ and $\mathcal{AC}$: they solve $\mathbb{G}$ in $\mathcal{ST}$ rings, and the near-gathering with termination in $\mathcal{AC}$ rings. However, if we consider $\mathcal{BRE}$, $\mathcal{RE}$ or $\mathcal{COT}$ rings, none of their algorithms succeeds to solve $\mathbb{G}$ or any non-trivial weaker version of $\mathbb{G}$. Indeed, while executing their algorithms in $\mathcal{BRE}$, $\mathcal{RE}$ or $\mathcal{COT}$ rings, each robot may stop its execution alone on a node. Therefore, if one or multiple classes of dynamic graphs among $\mathcal{BRE}$, $\mathcal{RE}$ and $\mathcal{COT}$ graphs are considered, their algorithms are not gracefully degrading anymore.

Algorithm proposed by Ooshita and Datta [128]: Their algorithm is gracefully degrading with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$: it solves $\mathbb{G}$ in $\mathcal{ST}$ and $\mathcal{BRE}$ rings, a weaker version of the gathering problem where all the robots have to terminate their execution in finite (but not necessarily bounded) time on the same node of the ring in $\mathcal{RE}$ rings, and the near-gathering without termination in $\mathcal{COT}$ and $\mathcal{AC}$ rings.

By the previous analysis, we can conclude that only two articles of the state of the art propose gracefully degrading gathering algorithms with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$.

In distributed computing, specifications are often described thanks to a safety and a liveness property. Intuitively, according to Lamport [117] "a safety property is one which states that something will not happen" and "a liveness property is one which states that something must happen".

The safety and liveness of $\mathbb{G}$ are defined as follows:

Safety: All robots that terminate do so on the same node.

Liveness: Every robot terminates in finite and bounded time.

The underlying idea of indulgent approaches [5, 118] is to preserve the safety of the problem even when the environment becomes harsher in order to preserve the essence of the problem.

We think that gracefully degrading algorithms should also follow this idea, and hence preserve the safety of the problem even when the environment becomes highly dynamic.

Note that, even if the algorithms of Di Luna et al. [122] are gracefully degrading with respect to $\mathcal{ST}$ and $\mathcal{AC}$ graphs, the version of the gathering problem solved in $\mathcal{AC}$ rings does not preserve the safety of the gathering problem (since the robots may terminate their execution on two different nodes). Moreover, even if the algorithm of Ooshita and Datta [128] is gracefully degrading with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$, it preserves the safety of the gathering problem only in a trivial way: there is no termination of the robots in $\mathcal{AC}$ and $\mathcal{COT}$ rings. Similarly, the algorithms of Izumi et al. [110] are gracefully degrading with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$, but there is no guarantee on the number of robots that must terminate their execution in $\mathcal{AC}$ and $\mathcal{COT}$ rings. In particular, it is possible for none of the robots to terminate their execution in $\mathcal{AC}$ and $\mathcal{COT}$ rings, and therefore, in this case, the safety of the gathering problem is only preserved trivially.

Therefore, there is no algorithm in the state of the art solving the gathering problem in a gracefully degrading manner with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$, considering weaker versions of the gathering problem that only weaken the liveness of this problem, and that guarantee that a certain number of robots must terminate their execution.

To fill the lack of the state of the art, we decide to conceive such an algorithm. We define specifications close to $\mathbb{G}$ that keep unchanged the safety property of $\mathbb{G}$, that weaken only its liveness and that impose that a certain number of the robots will terminate their execution. The liveness is weakened such that at most one robot may not terminate or (not exclusively) all robots

that terminate do so eventually (and not in a bounded time as in $\mathbb{G}$), giving us the three following specifications.

Definition 5.3 (Eventual gathering $\mathbb{G}_E$)**.** *All robots terminate their execution on the same node of the graph in finite time.*

Note that the impossibility result of Di Luna et al. [122] is also true for $\mathbb{G}_E$ (since $\mathbb{G}_E$ is weaker than $\mathbb{G}$): $\mathbb{G}_E$ is impossible in $\mathcal{AC}$ rings.

Definition 5.4 (Weak gathering $\mathbb{G}_W$)**.** *All robots but (at most) one terminate their execution on the same node of the graph in finite and bounded time.*

Definition 5.5 (Eventual weak gathering $\mathbb{G}_{EW}$)**.** *All robots but (at most) one terminate their execution on the same node of the graph in finite time.*

As indicated, $\mathbb{G}_E$, $\mathbb{G}_W$ and $\mathbb{G}_{EW}$ are weaker than $\mathbb{G}$. Moreover $\mathbb{G}_{EW}$ is weaker than $\mathbb{G}_E$ and than $\mathbb{G}_W$. However, $\mathbb{G}_E$ and $\mathbb{G}_W$ are not comparable. Indeed, in an execution where $\mathbb{G}_E$ is satisfied, it is possible to have all the robots that terminate their execution in a finite time that is arbitrarily long. Therefore, such execution does not satisfy $\mathbb{G}_W$. Similarly, in an execution where $\mathbb{G}_W$ is satisfied, it is possible to have all robots but one that terminate their execution in finite and bounded time, and to have one robot that never terminates its execution. Hence, such execution does not satisfy $\mathbb{G}_E$.

Chapter 6

Gathering

Contents

The gathering problem is impossible in $\mathcal{AC}$ rings [122]. This impossibility motivates the conception of a gracefully degrading gathering algorithm in dynamic rings (refer to Section 5.1). The conception of a gracefully degrading gathering algorithm implies to define at least one approached specification related to the gathering problem. We defined, in Section 5.3, three specifications related to $\mathbb{G}$: $\mathbb{G}_E$, $\mathbb{G}_W$ and $\mathbb{G}_{EW}$ (refer to Section 5.3 for the definitions of these specifications). These three related specifications preserve the safety of $\mathbb{G}$ and guarantee that at least $\mathcal{R}-1$ robots of the system terminate (eventually or in bounded time depending on the variants) their execution on the same node of the ring. In this chapter, based on these four specifications of the gathering problem ($\mathbb{G}$, $\mathbb{G}_E$, $\mathbb{G}_W$ and $\mathbb{G}_{EW}$), we present a gracefully degrading gathering algorithm with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$. Our algorithm solves $\mathbb{G}$ in $\mathcal{BRE}$ and $\mathcal{ST}$ rings, $\mathbb{G}_E$ in $\mathcal{RE}$ rings, $\mathbb{G}_W$ in $\mathcal{AC}$ rings and $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. It is the first gracefully degrading algorithm applied to robot networks. Since it is only possible to know the class to which belongs a dynamic graph in a post mortem way, one of the difficulties of such algorithm is to succeed to solve the correct version of the gathering problem without being able to detect the nature of the dynamics of the graph in which it is executed. Among the classes of dynamic rings considered some may possess one eventual missing edge while some others do not, which adds difficulty in the way to deal with the gathering problem while considering the different kinds of classes of dynamic graphs. To solve the gathering problem, since the nodes of the rings are anonymous, our algorithm consists in electing one robot (thanks to its identifier) and then make the other robots join this elected robot. Because of the presence of at most one eventual missing edge, it may be impossible to make the $\mathcal{R}-1$ robots of the system join the elected robot.

The results presented in this chapter have been published in SSS 2018 [31].

Results. In this chapter, we give a gracefully degrading gathering algorithm for dynamic rings. This algorithm is motivated by a set of impossibility results that we present in Section 6.1 (see also Table 6.1 for a summary of these impossibility results). We then present, in Section 6.2, our gracefully degrading gathering algorithm, and in Section 6.3 its proof of correctness. Note that, for each class of dynamic rings we consider, our algorithm solves the strongest possible of our variants of the gathering problem (refer to Table 6.1). Hence, our algorithm satisfies more properties than the one required by the definition of a gracefully degrading algorithm: we will detail this in the midterm perspectives described in Chapter 7. Finally, Section 6.4 concludes the chapter.

	$\mathbb{G}$	$\mathbb{G}_E$	$\mathbb{G}_W$	$\mathbb{G}_{EW}$
$\mathcal{COT}$	Impossible (Cor. 6.2, 6.3)	Impossible (Cor. 6.1)	Impossible (Cor. 6.3)	Possible (Th. 6.2)
$\mathcal{AC}$	Impossible (Cor. 6.2)	Impossible (Th. 6.1)	Possible (Th. 6.4)	—
$\mathcal{RE}$	Impossible (Cor. 6.3)	Possible (Th. 6.3)	Impossible (Cor. 6.3)	—
$\mathcal{BRE}$	Possible (Th. 6.5)	—	—	—
$\mathcal{ST}$	Possible (Cor. 6.6)	—	—	—

Table 6.1: Summary of our results. The symbol — means that a stronger variant of the problem is already proved solvable under the dynamics assumption. Our algorithm is gracefully degrading since it solves each variant of the gathering problem as soon as dynamics assumptions allow it.

In this chapter, we consider a system of $\mathcal{R} \geq 4$ robots. In addition to the common assumptions made on robots all along the chapters of this thesis (see Section 3.2), each robot knows the size n of the ring and $\mathcal{R}$. Each robot r possesses a distinct (positive) integer identifier id_r. Initially, a robot only knows the value of its own identifier. The robots are able to communicate (by direct reading) the values of their variables to each other only when they are involved in the same tower. They are endowed with strong local multiplicity detection. Finally, all the robots have the same chirality. Call this model $\mathcal{M}_{gathering}$ (also refer to Figure 3.5). These assumptions are needed for our algorithm but their necessity is left as an open question here. Thanks to all these assumptions, our algorithm succeeds to handle any arbitrary initial positions of the robots. In particular some robots may start their execution from the same node. For the algorithm presented in this chapter, the variable *dir* may take the value $right$, $left$ or $\perp$.

Note that, even if we present a gracefully degrading gathering algorithm (gathering whose specification adapts itself to the dynamics of the graphs), the robots have no means to detect the dynamics of the graph.

6.1 Impossibility Results

In this section, we present the set of impossibility results summarized in Table 6.1. These results show that some variants of the gathering problem cannot be solved depending on the dynamics of the ring in which the robots evolve and hence motivate our gracefully degrading approach.

First, we prove in Theorem 6.1 that $\mathbb{G}_E$ (the eventual variant of the gathering problem) is impossible to solve in $\mathcal{AC}$ rings. Note that Di Luna et al. [122] provide a similar result but show it in an informal way only. Moreover, our result subsumes theirs since the considered models are different: we show that the result remains valid even if robots are identified, able to communicate, and not necessarily initially all scattered (other different assumptions exist between the two models but have no influence on our proof).

The proof of Theorem 6.1 relies on the generic framework introduced by Braud-Santoni et al. [34] (see Section 4.2 for more information about this framework).

Theorem 6.1. *There exists no deterministic algorithm under $\mathcal{M}_{gathering}$ that satisfies $\mathbb{G}_E$ in $\mathcal{AC}$ rings of size 4 or more using 4 robots or more.*

Proof. By contradiction, assume that there exists a deterministic algorithm $\mathcal{A}$ that satisfies $\mathbb{G}_E$ in any $\mathcal{AC}$ ring of size 4 or more using 4 robots or more. Let us choose arbitrarily two of these robots and denote them r_1 and r_2.

Note that $\mathcal{A}$ may allow the last robot to terminate only if it is co-located with all other robots (otherwise, we obtain a contradiction with the safety of $\mathbb{G}_E$). So, proving the existence of an execution of $\mathcal{A}$ in an $\mathcal{AC}$ ring where r_1 and r_2 are never co-located is sufficient to obtain a contradiction with the liveness property of $\mathbb{G}_E$ and to show the result. This is the goal of the remainder of the proof.

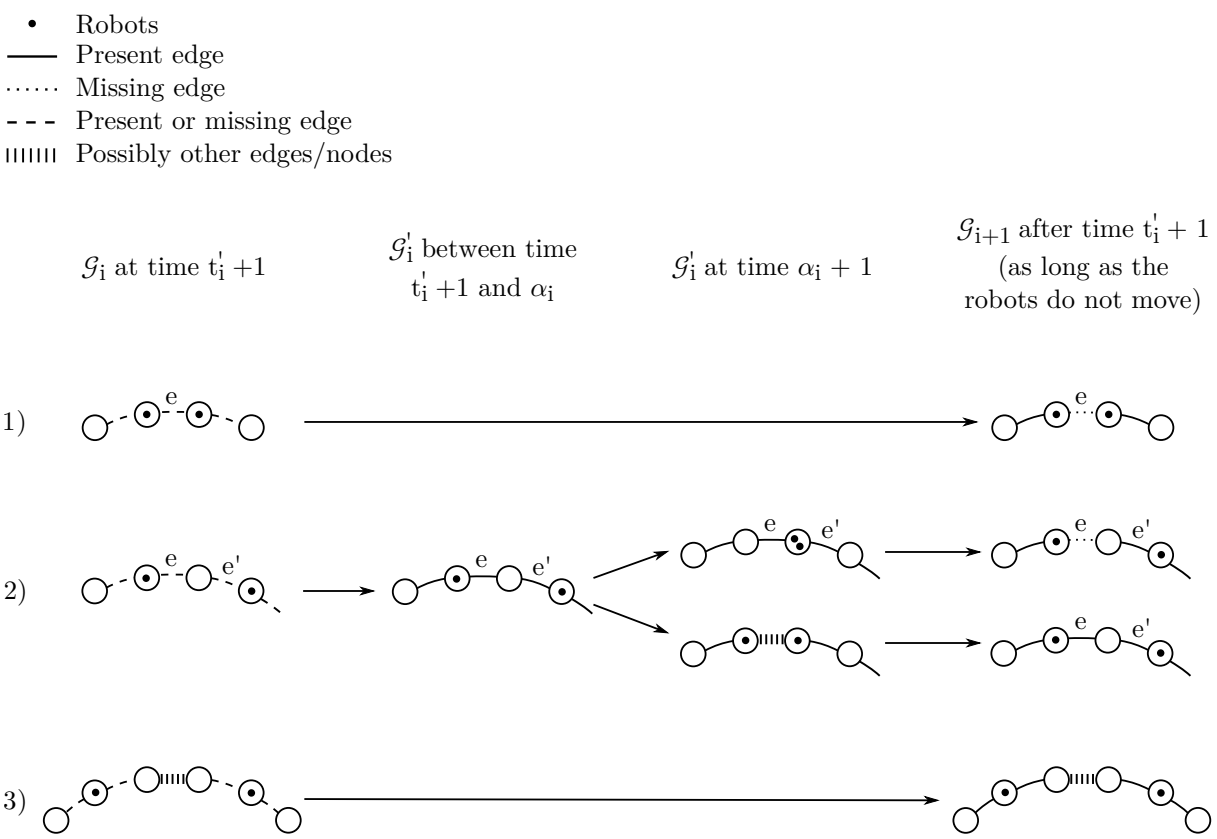


Figure 6.1: Construction of $\mathcal{G}_{i+1}$ from $\mathcal{G}_i$.

Let $\mathcal{G} = \{G_0, G_1, \ldots\}$ be an $\mathcal{AC}$ graph whose footprint G is a ring of size 4 or more such that $\forall i \in \mathbb{N}, G_i = G$. Consider two nodes u and v of $\mathcal{G}$, such that the node v is the adjacent node of u in G to the clockwise. We denote by e_{uv} the edge linking the nodes u and v. Let $\mathcal{G}'$ be $\mathcal{G}\backslash\{e_{uv}, \mathbb{N}\}$. Note that $\mathcal{G}' \in \mathcal{AC}$. Let ε be the execution of $\mathcal{A}$ in $\mathcal{G}'$ starting from the configuration where r_1 is located on node u and r_2 is located on node v. Note that the distance in the footprint of $\mathcal{G}$ between r_1 and r_2 (denoted $d(r_1, r_2)$) is equal to one.

Our goal is to construct a sequence of $\mathcal{AC}$ rings denoted $(\mathcal{G}_m)_{m\in\mathbb{N}}$ such that $\mathcal{G}_0 = \mathcal{G}'$ and, for any $i \geq 0$, r_1 and r_2 are never co-located before time t_i in ε_i (the execution of $\mathcal{A}$ in $\mathcal{G}_i$ starting from the same configuration as ε), $(t_m)_{m\in\mathbb{N}}$ being a strictly increasing sequence with $t_0 = 0$. First, we show in the next paragraph that, if some such $\mathcal{G}_i$ exists and, moreover, ensures the existence of a time $t'_i + 1 > t_i$ where the two robots are still on different nodes in ε_i, then we can construct $\mathcal{G}_{i+1}$ as shown in Figure 6.1. We then prove that our construction guarantees the existence of such a t'_i, implying the well definition of $(\mathcal{G}_m)_{m\in\mathbb{N}}$.

As r_1 and r_2 are not co-located at time t_i in ε_i, at least one of them must move in finite time in any execution starting from γ_{t_i} (otherwise, we obtain a contradiction with the liveness of $\mathbb{G}_E$). Let $t'_i \geq t_i$ be the smallest such time in the execution where the topology of the graph does not evolve from time t_i to time t'_i. In the following, we show how we construct the evolving graph $\mathcal{G}_{i+1}$, in function of t'_i and $\mathcal{G}_i$. As we assume that in $\mathcal{G}_i$, at time t'_i+1, r_1 and r_2 are on two different nodes, i.e., $d(r_1, r_2) \geq 1$, the following cases are possible.

Case 1: $d(r_1, r_2) = 1$ at time $t'_i + 1$.
Let e be the edge between the respective locations of r_1 and r_2 at time $t'_i + 1$. We define $\mathcal{G}_{i+1}$ on the same footprint as $\mathcal{G}_i$ by $\mathcal{G}_{i+1} = \mathcal{G}_i \otimes_{t'_i} (\mathcal{G}\backslash\{e, \{t'_i + 1, \ldots, +\infty\}\})$.

Case 2: $d(r_1, r_2) = 2$ at time $t'_i + 1$.

We define first $\mathcal{G}'_i$ on the same footprint as $\mathcal{G}_i$ by $\mathcal{G}'_i = \mathcal{G}_i \otimes_{t'_i} \mathcal{G}$. Note that $\mathcal{G}'_i$ is an $\mathcal{AC}$ ring by assumption on $\mathcal{G}_i$ and since $\mathcal{G}$ is the static ring. Then, to avoid a contradiction with the liveness of $\mathbb{G}_E$, we know that there exists a time $\alpha_i \geq t'_i + 1$ in the execution of $\mathcal{A}$ in $\mathcal{G}'_i$ where at least one of our two robots moves (*w.l.o.g.* assume that α_i is the smallest one). If, at time $\alpha_i + 1$, the two robots are on distinct nodes in $\mathcal{G}'_i$, then we define $\mathcal{G}_{i+1}$ on the same footprint as $\mathcal{G}_i$ by $\mathcal{G}_{i+1} = \mathcal{G}_i \otimes_{t'_i} \mathcal{G}$. If, at time $\alpha_i + 1$, the two robots are on the same node in $\mathcal{G}'_i$, then we denote e and e' the two consecutive edges between the respective locations of r_1 and r_2 that are crossed during the Move phase of time α_i, and we define $\mathcal{G}_{i+1}$ on the same footprint as $\mathcal{G}_i$ by $\mathcal{G}_{i+1} = \mathcal{G}_i \otimes_{t'_i} (\mathcal{G} \backslash \{e, \{t'_i + 1, \ldots, +\infty\}\})$.

Case 3: $d(r_1, r_2) > 2$ at time $t'_i + 1$.
We define $\mathcal{G}_{i+1}$ on the same footprint as $\mathcal{G}_i$ by $\mathcal{G}_{i+1} = \mathcal{G}_i \otimes_{t'_i} \mathcal{G}$.

Note that $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ are indistinguishable for robots until time t'_i. This implies that, at time $t'_i + 1$, r_1 and r_2 are on the same nodes in ε_i and in ε_{i+1}. By construction of t'_i, either r_1 or r_2 or both of the two robots move at time t'_i in ε_{i+1}. Moreover, by construction of $\mathcal{G}_i$, even if one or both of the robots move during the Move phase of time t'_i, at time $t'_i + 1$ the robots are still on two distinct nodes (since, in all cases above, either the distance between the robots before the move is strictly greater than 2, an edge between the two robots is missing before the move and prevents the meeting, or the two robots move in a way that prevents the meeting by indistinguishability between $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$). Note that, by construction, $\mathcal{G}_{i+1}$ has at most one edge missing at each instant time and hence is an $\mathcal{AC}$ ring.

Defining $t_{i+1} = t'_i + 1$, we succeed to construct $\mathcal{G}_{i+1}$ with the desired properties. Note that t'_i and $\mathcal{G}_0$ trivially satisfy all our assumptions. In other words, $(\mathcal{G}_m)_{m \in \mathbb{N}}$ is well-defined.

We can then define the evolving graph $\mathcal{G}_\omega$ such that $\mathcal{G}_\omega$ and $\mathcal{G}_0$ have the same footprint, and such that for all $i \in \mathbb{N}$, $\mathcal{G}_\omega$ shares a common prefix with $\mathcal{G}_i$ until time t'_i. As the sequence $(t_m)_{m \in \mathbb{N}}$ is increasing by construction, this implies that the sequence $(\mathcal{G}_m)_{m \in \mathbb{N}}$ converges to $\mathcal{G}_\omega$. Applying the theorem of Braud-Santoni et al. [34], we obtain that, until time t'_i, the execution of $\mathcal{A}$ in $\mathcal{G}_\omega$ is identical to the one in $\mathcal{G}_i$. This implies that, executing $\mathcal{A}$ in $\mathcal{G}_\omega$ (whose footprint is a ring of size 4 or more), r_1 and r_2 are always on distinct nodes, contradicting the liveness of $\mathbb{G}_E$ and proving the result. □

It is possible to derive some other impossibility results from Theorem 6.1. Indeed, by Corollary 4.1, the inclusion $\mathcal{AC} \subset \mathcal{COT}$ allows us to state that $\mathbb{G}_E$ is impossible to satisfy under $\mathcal{M}_{gathering}$ in $\mathcal{COT}$ rings as well.

Corollary 6.1. *There exists no deterministic algorithm under $\mathcal{M}_{gathering}$ that satisfies $\mathbb{G}_E$ in $\mathcal{COT}$ rings of size 4 or more using 4 robots or more.*

From the very definitions of $\mathbb{G}$ and $\mathbb{G}_E$, it is straightforward to see that the impossibility of $\mathbb{G}_E$ under a given class of dynamic rings implies the one of $\mathbb{G}$ under the same class of dynamic rings (refer to Section 4.1).

Corollary 6.2. *There exists no deterministic algorithm under $\mathcal{M}_{gathering}$ that satisfies $\mathbb{G}$ in $\mathcal{COT}$ or $\mathcal{AC}$ rings of size 4 or more using 4 robots or more.*

Finally, impossibility results for bounded variants of the gathering problem (i.e., the impossibility of $\mathbb{G}$ in $\mathcal{RE}$ rings and of $\mathbb{G}_W$ in $\mathcal{COT}$ and $\mathcal{RE}$ rings) are obtained as follows. The definition of $\mathcal{COT}$ and $\mathcal{RE}$ does not exclude the ability for all edges of the graph to be missing initially and for any arbitrary long time, hence preventing the gathering of robots for any arbitrary long time if they are initially scattered. This observation is sufficient to prove a contradiction with the existence of an algorithm solving $\mathbb{G}$ or $\mathbb{G}_W$ in these classes of dynamic rings.

Corollary 6.3. *There exists no deterministic algorithm under $\mathcal{M}_{gathering}$ that satisfies $\mathbb{G}$ or $\mathbb{G}_W$ in $\mathcal{COT}$ or $\mathcal{RE}$ rings of size 4 or more using 4 robots or more.*

6.2 Gracefully Degrading Gathering: Algorithm $\mathbb{GDG}$

This section presents $\mathbb{GDG}$, our gracefully degrading gathering algorithm, that aims to solve different variants of the gathering problem under various dynamics (refer to Table 6.1).

In Subsection 6.2.1, we informally describe our algorithm clarifying which variant of gathering is satisfied within which class of evolving graphs. Next, Subsection 6.2.2 presents formally the algorithm.

6.2.1 Overwiew

Our algorithm has to overcome various difficulties. First, robots are evolving in an environment in which no node can be distinguished. So, the trivial algorithm in which the robots meet on a particular node is impossible. Moreover, since the footprint of the graph is a ring, (at most) one of the n edges may be an eventual missing edge. Indeed, in the strongest class of dynamic rings considered ($\mathcal{COT}$ rings), at most one edge may be an eventual missing edge: if there were more than one eventual missing edge, then the eventual underlying graph of the dynamic ring would possess at least two distinct connected components and it would not be possible for a node to reach over time any other node infinitely often (which is the property that $\mathcal{COT}$ graphs should respect). No robot is able to distinguish an eventual missing edge from a missing edge that will appear later in the execution. In particular, a robot stuck by a missing edge does not know whether it can wait for the missing edge to appear again or not. Finally, despite the fact that no robot is aware of which class of dynamic graphs robots are evolving in, the algorithm is required to meet at least the specification of the gathering according to the class of dynamic graphs in which it is executed or a better specification than this one.

The overall scheme of the algorithm consists in first detecting r_{min}, the robot having the minimum identifier so that the $\mathcal{R}$ robots eventually gather on its node (i.e., satisfying specification $\mathbb{G}_E$). Of course, depending on the particular evolving graph in which our algorithm is executed, $\mathbb{G}_E$ may not achieve. In the strongest class of evolving graphs considered in this chapter (class $\mathcal{COT}$) and the "worst" possible evolving graph, one can expect specification $\mathbb{G}_{EW}$ only, i.e., at least $\mathcal{R}-1$ robots gathered.

The algorithm proceeds in four successive phases: `M` (for "am I the Min?"), `K` (for "min wait to be Known"), `W` (for "Walk"), and `T` (for "wait Termination"). Actually, again depending on the class of graphs and the evolving graph in which our algorithm is executed, we will see that the four phases are not necessarily all executed since the execution can be stopped prematurely, especially in case where $\mathbb{G}_E$ (or $\mathbb{G}$) is achieved. By contrast, they can also never be completed in some strong classes of dynamic graphs where the connectivity assumptions are weak (namely $\mathcal{AC}$ or $\mathcal{COT}$ graphs), solving $\mathbb{G}_{EW}$ (or $\mathbb{G}_W$) only.

We present below the intuition of each of these four phases in the order.

Phase `M`. This phase leads each robot to know whether it possesses the minimum identifier. Initially every robot r points to the *right* direction. Then r moves to the *right* until it moves $4*n*id_r$ steps on the right (where id_r is the identifier of r, and n is the size of the ring) or until it meets $\mathcal{R}-2$ other robots such that its identifier is not the smaller one among these robots or until it meets a robot that knows the identifier of r_{min}. The first robot that succeeds to move $4*n*id_r$ steps in the right direction is necessarily r_{min}. Depending on the class of graph, one eventual missing edge may exist, preventing r_{min} to move on the *right* direction during $4*n*id_{r_{min}}$ steps.

However, in the case where there is an eventual missing edge at least $\mathcal{R}-1$ robots succeed to be located on a same node. They are located either on the extremity of the eventual missing edge or on the extremity of a missing edge that is not eventually missing. The robot r_{min} is not necessarily located with these $\mathcal{R}-1$ gathered robots. Note that the weak form of gathering ($\mathbb{G}_{EW}$) could be solved in that case. However, the $\mathcal{R}-1$ robots gathered cannot stop their execution. Indeed, our algorithm aims at gathering the robots on the node occupied by r_{min}. However, r_{min} may not be part of the $\mathcal{R}-1$ robots that gathered. Further, it is possible for $\mathcal{R}-1$ robots to

Algorithm 6.1 Predicates used in $\mathbb{GDG}$

MinDiscovery() $\equiv$
$(kind_r = potentialMin \wedge \exists r' \in NodeMate(), (kind_{r'} = righter \wedge id_r < id_{r'}))\vee$
$\exists r' \in NodeMate(), idMin_{r'} = id_r \vee$
$\exists r' \in NodeMate(), (kind_{r'} \in \{dumbSearcher, potentialMin\} \wedge id_r < idPotentialMin_{r'})\vee$
$rightSteps_r = 4 * id_r * n$

GE() $\equiv$
$|NodeMate()| = \mathcal{R} - 1$

GEW() $\equiv$
$|NodeMate()| = \mathcal{R} - 2 \wedge \exists r' \in \{r\} \cup NodeMate(), kind_{r'} \in \{minWaitingWalker, minTailWalker\}$

HeadWalkerWithoutWalkerMate() $\equiv$
$kind_r = headWalker \wedge ExistsEdge(left, previous) \wedge \neg HasMoved() \wedge NodeMateIds() \neq walkerMate_r$

LeftWalker() $\equiv$
$kind_r = leftWalker$

HeadOrTailWalkerEndDiscovery() $\equiv$
$kind_r \in \{headWalker, tailWalker, minTailWalker\} \wedge walkSteps_r = n$

HeadOrTailWalker() $\equiv$
$kind_r \in \{headWalker, tailWalker, minTailWalker\}$

AllButTwoWaitingWalker() $\equiv$
$|NodeMate()| = \mathcal{R} - 3 \wedge \forall r' \in \{r\} \cup NodeMate(), kind_{r'} \in \{waitingWalker, minWaitingWalker\}$

WaitingWalker() $\equiv$
$kind_r \in \{waitingWalker, minWaitingWalker\}$

PotentialMinOrSearcherWithMinWaiting(r') $\equiv$
$kind_r \in \{potentialMin, dumbSearcher, awareSearcher\} \wedge kind_{r'} = minWaitingWalker$

RighterWithMinWaiting(r') $\equiv$
$kind_r = righter \wedge kind_{r'} = minWaitingWalker$

NotWalkerWithHeadWalker(r') $\equiv$
$kind_r \in \{righter, potentialMin, dumbSearcher, awareSearcher\} \wedge kind_{r'} = headWalker$

NotWalkerWithTailWalker(r') $\equiv$
$kind_r \in \{righter, potentialMin, dumbSearcher, awareSearcher\} \wedge kind_{r'} = minTailWalker$

PotentialMinWithAwareSearcher(r') $\equiv$
$kind_r = potentialMin \wedge kind_{r'} = awareSearcher$

AllButOneRighter() $\equiv$
$|NodeMate()| = \mathcal{R} - 2 \wedge \forall r' \in \{r\} \cup NodeMate(), kind_{r'} = righter$

RighterWithSearcher(r') $\equiv$
$kind_r = righter \wedge kind_{r'} \in \{dumbSearcher, awareSearcher\}$

PotentialMinOrRighter() $\equiv$
$kind_r \in \{potentialMin, righter\}$

DumbSearcherMinRevelation() $\equiv$
$kind_r = dumbSearcher \wedge \exists r' \in NodeMate(), (kind_{r'} = righter \wedge id_{r'} > idPotentialMin_r)$

DumbSearcherWithAwareSearcher(r') $\equiv$
$kind_r = dumbSearcher \wedge kind_{r'} = awareSearcher$

Searcher() $\equiv$
$kind_r \in \{dumbSearcher, awareSearcher\}$

gather (without r_{min}) even when r_{min} succeeds in moving $4*n*id_{r_{min}}$ steps to the right (i.e., even when r_{min} stops to move because it completed Phase `M`). In that case, if the $\mathcal{R}-1$ robots that gathered stop their execution, $\mathbb{G}_E$ cannot be solved in $\mathcal{RE}$, $\mathcal{BRE}$ and $\mathcal{ST}$ rings, as $\mathbb{GDG}$ should do. Note that, it is also possible for r_{min} to be part of the $\mathcal{R}-1$ robots that gathered.

Recall that robots can communicate when they are both located in the same node. So, the $\mathcal{R}-1$ robots may be aware of the identifier of the robot with the minimum identifier among them. Since it can or cannot be the actual r_{min}, let us call this robot *potentialMin*. Then, driven by *potentialMin*, a search phase starts during which the $\mathcal{R}-1$ robots try to visit all the nodes of the ring infinitely often in both directions by subtle round trips. Doing so, r_{min} eventually knows that it possesses the actual minimum identifier.

Phase `K`**.** The goal of the second phase consists in spreading the identifier of r_{min} among the other robots. The basic idea is that during this phase, r_{min} stops moving and waits until $\mathcal{R}-3$ other robots join it on its node so that its identifier is known by at least $\mathcal{R}-3$ other robots. The obvious question arises:"*Why waiting for* $\mathcal{R}-3$ *extra robots only?*" A basic idea to gather

could be that once r_{min} is aware that it possesses the minimum identifier, it can just stop to move and just wait for the other robots to eventually reach its location, just by moving toward the right direction. Actually, depending on the particular evolving graph considered one missing edge e may eventually appear, preventing robots from reaching r_{min} by moving toward the same direction only. That is why the gathering of the $\mathcal{R}-2$ robots is eventually achieved by the same search phase as in Phase `M` (since the search phase permits to at least 3 robots to explore infinitely often the nodes of the ring until reaching a given node). However, by doing this, it is possible to have 2 robots stuck on each extremity of e. Further, these two robots cannot change the directions they point to since a robot is not able to distinguish an eventual missing edge from a missing edge that will appear again later. This is why during Phase `K`, r_{min} stops to move until $\mathcal{R}-3$ other robots join it to form a tower of $\mathcal{R}-2$ robots. In this way, these $\mathcal{R}-2$ robots start the third phase simultaneously.

Phase `W`. The third phase is a *walk* made by the tower of $\mathcal{R}-2$ robots. The $\mathcal{R}-2$ robots are split into two distinct groups, *Head* and *Tail*. Head is the unique robot with the maximum identifier of the tower. Tail, composed of $\mathcal{R}-3$ robots, is made of the other robots of the tower, led by r_{min}. Both move alternatively in the *right* direction during n steps such that between two movements of a given group the two groups are again located on a same node. This movement permits to prevent the two robots that do not belong to any of these two groups to be both stuck on different extremities of an eventual missing edge (if any) once this walk is finished. Since there exists at most one eventual missing edge, we are sure that if the robots that have executed the walk stop moving forever, then at least one robot can join them during the next and last phase.

As noted, it can exist an eventual missing edge, therefore, Head and Tail may not complete Phase `W`. Indeed, one of the two situations below may occur: (i) Head and Tail together form a tower of $\mathcal{R}-2$ robots but an eventual missing edge on their right prevents them to complete Phase `W`; (ii) Head and Tail are located on neighboring nodes and the edge between them is an eventual missing edge that prevents Head and Tail to continue to move alternatively.

Call u the node where Tail is stuck on an eventual missing edge. In the two situations described even if Phase `W` is not complete by both Head and Tail, either $\mathbb{G}_E$ or $\mathbb{G}_{EW}$ is solved. Indeed, in the first situation, necessarily at least one robot r succeeds to join u. In fact, either r points to the good direction to reach u or it meets a robot on the other extremity of the eventual missing edge that makes it change its direction, and hence makes it point to the good direction to reach u. In the second situation, necessarily at least two robots r and r' succeed to join u. This is done either because r and r' point to the good direction to reach u or because they reach the node where Head is located without Tail making them change their direction, and hence making them point to the good direction to reach u.

Once a tower of $\mathcal{R}-1$ robots including r_{min} is formed, $\mathbb{G}_{EW}$ is solved. Then, the latter robot tries to reach the tower to eventually solve $\mathbb{G}_E$ in favorable cases.

Phase `T`. The last phase starts once the robots of Head have completed Phase `W`. If it exists a time at which the robots of Tail complete Phase `W`, then Head and Tail form a tower of $\mathcal{R}-2$ robots and stop moving. As explained in the previous phase, Phase `W` ensures that at least one extra robot eventually joins the node where Head and Tail are located to form a tower of $\mathcal{R}-1$ robots. Once a tower of $\mathcal{R}-1$ robots including r_{min} is formed, $\mathbb{G}_{EW}$ is solved. Then, the latter robot tries to reach the tower to eventually solve $\mathbb{G}_E$ in favorable cases. In the case the robots of Tail never complete the phase `W`, then this implies that Head and Tail are located on neighboring nodes and that the edge between them is an eventual missing edge. As described in Phase `W` either $\mathbb{G}_{EW}$ or $\mathbb{G}_E$ is solved.

6.2.2 Algorithm

Before presenting formally our algorithm, we first describe the set of variables of each robot. Similarly as for the variables declared in Section 3.2, when necessary, we index the variables with

Algorithm 6.2 Functions used in $\mathbb{GDG}$

Function StopMoving()

$dir_r := \bot$

Function MoveLeft()

$dir_r := left$

Function BecomeLeftWalker()

$(kind_r, dir_r) := (leftWalker, \bot)$

Function Walk()

$$dir_r := \begin{cases} \bot & \text{if } (id_r = idHeadWalker_r \wedge walkerMate_r \neq NodeMateIds()) \vee \\ & \quad (id_r \neq idHeadWalker_r \wedge idHeadWalker_r \in NodeMateIds()) \\ right & \text{otherwise} \end{cases}$$

$walkSteps_r := walkSteps_r + 1$ if $dir_r = right \wedge ExistsEdge(right, current)$

Function InitiateWalk()

$idHeadWalker_r := \max(\{id_r\} \cup NodeMateIds())$

$walkerMate_r := NodeMateIds()$

$$kind_r := \begin{cases} headWalker & \text{if } id_r = idHeadWalker_r \\ minTailWalker & \text{if } kind_r = minWaitingWalker \\ tailWalker & \text{otherwise} \end{cases}$$

Function BecomeWaitingWalker(r')

$(kind_r, idPotentialMin_r, idMin_r, dir_r) := (waitingWalker, id_{r'}, id_{r'}, \bot)$

Function BecomeMinWaitingWalker()

$(kind_r, idPotentialMin_r, idMin_r, dir_r) := (minWaitingWalker, id_r, id_r, \bot)$

Function BecomeAwareSearcher(r')

$(kind_r, dir_r) := (awareSearcher, right)$

$$(idPotentialMin_r, idMin_r) := \begin{cases} (idPotentialMin_{r'}, idPotentialMin_{r'}) & \text{if } kind_{r'} = dumbSearcher \\ (idMin_{r'}, idMin_{r'}) & \text{otherwise} \end{cases}$$

Function BecomeTailWalker(r')

$(kind_r, idPotentialMin_r, idMin_r) := (tailWalker, idPotentialMin_{r'}, idMin_{r'})$

$(idHeadWalker_r, walkerMate_r, walkSteps_r) := (idHeadWalker_{r'}, walkerMate_{r'}, walkSteps_{r'})$

Function MoveRight()

$dir_r := right$

$rightSteps_r := rightSteps_r + 1$ if $ExistsEdge(dir, current)$

Function InitiateSearch()

$idPotentialMin_r := \min(\{id_r\} \cup NodeMateIds())$

$$kind_r := \begin{cases} potentialMin & \text{if } id_r = idPotentialMin_r \\ dumbSearcher & \text{otherwise} \end{cases}$$

$rightSteps_r := rightSteps_r + 1$ if $kind_r = potentialMin \wedge ExistsEdge(dir, current)$

Function Search()

$$dir_r := \begin{cases} left & \text{if } |NodeMate()| \geq 1 \wedge id_r = \max(\{id_r\} \cup NodeMateIds()) \\ right & \text{if } |NodeMate()| \geq 1 \wedge id_r \neq \max(\{id_r\} \cup NodeMateIds()) \\ dir_r & \text{otherwise} \end{cases}$$

the name of the robot to clarify to which robot the variable belongs to. We recall that each robot knows $\mathcal{R}$, n and its own identifier id (strictly greater than 0) as constants.

In addition to the variable dir_r (initialized to $right$), each robot r possesses seven variables described below. Variable $kind_r$ allows the robot r to know which phase of the algorithm it is performing and (partially) indicates which movement the robot has to execute. The possible values for this variable are $righter$, $dumbSearcher$, $awareSearcher$, $potentialMin$, $waitingWalker$, $minWaitingWalker$, $headWalker$, $tailWalker$, $minTailWalker$ and $leftWalker$. Initially, the variable $kind_r$ is equal to $righter$. Initialized with 0, $rightSteps_r$ counts the number of steps done by r in the $right$ direction when $kind_r \in \{righter, potentialMin\}$. The next variable is $idPotentialMin_r$. Initially equals to -1, the variable $idPotentialMin_r$ contains the identifier of

Algorithm 6.3 $\mathbb{GDG}$

Rules for Termination

$\mathbf{Term_1} :: \mathbb{G}_E() \longrightarrow$ **terminate**
$\mathbf{Term_2} :: \mathbb{G}_{EW}() \longrightarrow$ **terminate**

Rules for Phase `T`

$\mathbf{T_1} :: LeftWalker() \longrightarrow$ MoveLeft()
$\mathbf{T_2} :: HeadWalkerWithoutWalkerMate() \longrightarrow$ BecomeLeftWalker()
$\mathbf{T_3} :: HeadOrTailWalkerEndDiscovery() \longrightarrow$ StopMoving()

Rules for Phase `W`

$\mathbf{W_1} :: HeadOrTailWalker() \longrightarrow$ Walk()

Rules for Phase `K`

$\mathbf{K_1} :: AllButTwoWaitingWalker() \longrightarrow$ InitiateWalk()
$\mathbf{K_2} :: WaitingWalker() \longrightarrow$ StopMoving()
$\mathbf{K_3} :: \exists r' \in NodeMate(), PotentialMinOrSearcherWithMinWaiting(r')$
$\longrightarrow$ BecomeWaitingWalker(r')
$\mathbf{K_4} :: \exists r' \in NodeMate(), RighterWithMinWaiting(r') \wedge ExistsEdge(right, current)$
$\longrightarrow$ BecomeAwareSearcher(r')

Rules for Phase `M`

$\mathbf{M_1} :: PotentialMinOrRighter() \wedge MinDiscovery() \longrightarrow$ BecomeMinWaitingWalker(r)
$\mathbf{M_2} :: \exists r' \in NodeMate(), NotWalkerWithHeadWalker(r') \wedge ExistsEdge(right, current)$
$\longrightarrow$ BecomeAwareSearcher(r')
$\mathbf{M_3} :: \exists r' \in NodeMate(), NotWalkerWithHeadWalker(r')$
$\longrightarrow$ BecomeAwareSearcher(r'); StopMoving()
$\mathbf{M_4} :: \exists r' \in NodeMate(), NotWalkerWithTailWalker(r') \longrightarrow$ BecomeTailWalker(r'); Walk()
$\mathbf{M_5} :: \exists r' \in NodeMate(), PotentialMinWithAwareSearcher(r')$
$\longrightarrow$ BecomeAwareSearcher(r'); Search()
$\mathbf{M_6} :: AllButOneRighter() \longrightarrow$ InitiateSearch()
$\mathbf{M_7} :: \exists r' \in NodeMate(), RighterWithSearcher(r') \longrightarrow$ BecomeAwareSearcher(r'); Search()
$\mathbf{M_8} :: PotentialMinOrRighter() \longrightarrow$ MoveRight()
$\mathbf{M_9} :: DumbSearcherMinRevelation() \longrightarrow$ BecomeAwareSearcher(r); Search()
$\mathbf{M_{10}} :: \exists r' \in NodeMate(), DumbSearcherWithAwareSearcher(r')$
$\longrightarrow$ BecomeAwareSearcher(r'); Search()
$\mathbf{M_{11}} :: Searcher() \longrightarrow$ Search()

the robot that possibly possesses the minimum identifier (a positive integer) of the system. This variable is especially set when $\mathcal{R}-1$ *righter* are located on a same node. In this case, the variable $idPotentialMin_r$ of each robot r that is involved in the tower of $\mathcal{R}-1$ robots is set to the value of the minimum identifier possessed by these robots. The variable $idMin_r$ of a robot indicates the identifier of the robot that possesses the actual minimum identifier among all the robots of the system. This variable is initially set to -1. Let the variable $walkerMate_r$ of a robot be the set of all the identifiers of the $\mathcal{R}-2$ robots that initiate the Phase `W`. Initially this variable is set to $\emptyset$. The counter $walkSteps_r$, initially 0, maintains the number of steps done in the right direction while r performs the Phase `W`. Finally, the variable $idHeadWalker_r$ contains the identifier of the robot that plays the part of Head during the Phase `W`.

Moreover, we assume the existence of a specific instruction: **terminate**. By executing this instruction, a robot stops executing the L-C-M cycles forever.

To ease the writing of our algorithm, we define a set of predicates (presented in Algorithm 6.1) and functions (presented in Algorithm 6.2), that are used in our gracefully degrading algorithm $\mathbb{GDG}$ (refer to Algorithm 6.3).

6.3 Proofs of correctness of $\mathbb{GDG}$

In this section, we first prove, in Section 6.3.1, that $\mathbb{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. Then, in Section 6.3.2, we consider $\mathcal{AC}$, $\mathcal{RE}$, $\mathcal{BRE}$ and $\mathcal{ST}$ rings and for each of these classes of dynamic

rings, we give the problem $\mathbb{GDG}$ solves in it.

We want to prove that, while executing $\mathbb{GDG}$, at least $\mathcal{R}-1$ robots terminate their execution on the same node. Therefore, in the proof of correctness, we show that our algorithm forces the robots to execute either Rule $\mathbf{Term_1}$ or Rule $\mathbf{Term_2}$ whatever the harsh situation. More precisely, the logic of our proof of correctness is based on a proof by contradiction: in each proof, we make the assumption that Rules $\mathbf{Term_1}$ and $\mathbf{Term_2}$ are not executed and then finally we prove that our algorithm succeeds to execute at least one of these rules. Note that this assumption is made overall the proof of correctness and hence we will not indicate it at the beginning of each proof.

In the following, for ease of reading, we abuse the various values of the variable *kind* to qualify the robots. For instance, if the current value of variable *kind* of a robot is *righter*, then we say that the robot is a *righter* robot. Let us call *min* a robot such that its variable *kind* is equal either to $minWaitingWalker$ or to $minTailWalker$. Moreover, in some proofs, we indicate the functions called by the robots. When these functions take a parameter and the value of this parameter does not matter for the proof, we simply symbolize it by the character "_". For instance, if a robot calls the function BecomeAwareSearcher, we write that it calls the function BecomeAwareSearcher(_).

6.3.1 $\mathbb{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings

In this section, we prove that $\mathbb{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. Since $\mathbb{GDG}$ is divided into four phases, we prove each of these phases hereafter.

Correctness Proof of Phase `M`

We recall that the goal of Phase `M` of our algorithm is to make the robot with the minimum identifier aware that it possesses the minimum identifier among all the robots of the system. In our algorithm a robot is aware that it possesses the minimum identifier when it is *min*. Therefore, in this section we have to prove that only r_{min} can become *min*, and that r_{min} effectively becomes *min* in finite time. We prove this respectively in Lemmas 6.3 and 6.5.

First we give two observations that help us all along the proves of each phase.

Observation 6.1. *By the rules of $\mathbb{GDG}$, a robot whose variable kind is not either righter or $potentialMin$ cannot become a righter or a $potentialMin$.*

Observation 6.2. *By the rules of $\mathbb{GDG}$, a robot whose variable kind is not righter cannot become a righter robot.*

While executing $\mathbb{GDG}$, once a robot knows that it possesses the minimum identifier, it remembers this information. In other words, once a robot becomes *min* it stays *min* during the rest of the execution. We prove this statement in the following lemma.

Lemma 6.1. *min is a closed state under $\mathbb{GDG}$.*

Proof. A robot is a *min* when its variable *kind* is either equal to $minWaitingWalker$ or to $minTailWalker$. A $minTailWalker$ robot can only execute the rules $\mathbf{T_3}$ and $\mathbf{W_1}$ that do not update the variable *kind*. A $minWaitingWalker$ robot can only execute the rules $\mathbf{K_1}$ and $\mathbf{K_2}$ that respectively makes it become a $minTailWalker$ and does not change its state. □

In the following lemma, we prove that *righter* and $potentialMin$ are robots that always point to the right direction. This lemma helps us to prove the correctness of Phase `M`, as well as the correctness of Phase `K`.

Lemma 6.2. *If, at a time t, a robot is a righter or a $potentialMin$, then it points to the right direction from the beginning of the execution until the Look phase of time t.*

Proof. Robots that are *righter* robots in a configuration γ_i at time i and that are still *righter* in the configuration γ_{i+1}, point to the *right* direction during the move Phase of time i (Rule $\mathbf{M_8}$). Moreover, by Observation 6.2 and since initially all the robots are *righter* robots and point to the *right* direction, if a robot is a *righter* during the Look phase of a time t, this implies that it points to the *right* direction from the beginning of the execution until the Look phase of time t.

Similarly, robots that are $potentialMin$ robots in a configuration γ_i at time i and that are still $potentialMin$ in the configuration γ_{i+1}, point to the *right* direction during the move Phase of time i (Rule $\mathbf{M_8}$). The only way for a robot to become a $potentialMin$ is to be a *righter* and to execute Rule $\mathbf{M_6}$. While executing Rule $\mathbf{M_6}$, a *righter* that becomes $potentialMin$ does not change the direction it points to. Therefore, by Observations 6.1 and 6.2, and by the arguments of the first paragraph, this implies that if a robot is a $potentialMin$ during the Look phase of a time t, then it points to the *right* direction from the beginning of the execution until the Look phase of time t. □

Now we prove one of the two main lemmas of this phase: we prove that only r_{min} can be aware that it possesses the minimum identifier among all the robots of the system.

Lemma 6.3. *Only r_{min} can become min.*

Proof. Assume that there exists a robot $r \neq r_{min}$ that becomes min. Assume also that r is the first robot different from r_{min} that becomes min. By definition of r_{min}, $id_r > id_{r_{min}}$.

A robot that is a min is a robot such that its variable *kind* is either equal to $minWaiting-Walker$ or to $minTailWalker$. A robot becomes $minTailWalker$ only if it executes Rule $\mathbf{K_1}$. A robot can execute Rule $\mathbf{K_1}$ only if it is a $minWaitingWalker$. A robot becomes $minWaiting-Walker$ only if it executes Rule $\mathbf{M_1}$. Only *righter* robots or $potentialMin$ robots can execute Rule $\mathbf{M_1}$ (refer to predicate $PotentialMinOrRighter()$). Then by Observation 6.1, we conclude that each robot can execute Rule $\mathbf{M_1}$ at most once. $(*)$

In the following, let us consider the different conditions of the predicate $MinDiscovery()$ of Rule $\mathbf{M_1}$ that permits r to become min.

Case 1: r becomes min because the condition "$\mathbf{kind_r = potentialMin \wedge \exists r' \in Node-Mate()}$, $\mathbf{(kind_{r'} = righter \wedge id_r < id_{r'})}$" is true.

The only way for a robot to have its variable *kind* set to $potentialMin$ is to execute Rule $\mathbf{M_6}$. This rule is executed when $\mathcal{R} - 1$ *righter* robots are on a same node. Among these $\mathcal{R}-1$ *righter* robots, the one with the minimum identifier sets its variable *kind* to $potential-Min$ while the other robots set their variables *kind* to *dumbSearcher*. By Observation 6.1, a robot that becomes a *dumbSearcher* robot after the execution of Rule $\mathbf{M_6}$ can never become *righter* robot or $potentialMin$ robot. Moreover, by Observation 6.2, a robot that becomes a $potentialMin$ can never become a *righter*. Since $\mathcal{R} - 1$ *righter* are needed to execute Rule $\mathbf{M_6}$, this rule can be executed only once during the execution. Therefore if r is a $potentialMin$, it is necessarily the robot that possesses the minimum identifier among the $\mathcal{R} - 1$ robots that execute Rule $\mathbf{M_6}$. Moreover, if there exists a *righter* robot r' when r is $potentialMin$, this implies that r' has not executed Rule $\mathbf{M_6}$. Hence if $id_r < id_{r'}$, this necessarily implies that $r = r_{min}$, therefore there is a contradiction with the fact that $r \neq r_{min}$.

Case 2: r becomes min because the condition "$\mathbf{\exists r' \in NodeMate()}$, $\mathbf{idMin_{r'} = id_r}$" is true.

By $(*)$, r is not yet min at the time of its meeting with r'. A robot r' can update its variable $idMin$ with the identifier (other than its) of a robot that is not min only when it executes Rules $\mathbf{M_5}$, $\mathbf{M_7}$, $\mathbf{M_9}$ or $\mathbf{M_{10}}$. Among these rules only the rules $\mathbf{M_7}$ (in the case a *righter* is located with a *dumbSearcher*) and $\mathbf{M_9}$ permit a robot to update its variable $idMin$ with the identifier of a robot without copying the value of the variable $idMin$ of another robot.

Therefore at least one of these rules is necessarily executed at a time, since initially the variables $idMin$ of the robots are equal to -1. To execute Rule $\mathbf{M_7}$ (in the case a *righter* is located with a *dumbSearcher*) or Rule $\mathbf{M_9}$, a *dumbSearcher* robot must be present in the execution. Only the execution of Rule $\mathbf{M_6}$ permits to have *dumbSearcher* robots in the execution. This rule is executed when $\mathcal{R}-1$ *righter* robots are on a same node. The $\mathcal{R}-1$ robots that execute this rule, set their variables $idPotentialMin$ to the identifier of the robot that becomes $potentialMin$ while executing this rule. Moreover if a robot is a *dumbSearcher* in a configuration γ_t at time t and is still a *dumbSearcher* in the configuration γ_{t+1} then it does not update its variable $idPotentialMin$ during time t (since it executes Rule $\mathbf{M_{11}}$).

In the case Rule $\mathbf{M_7}$ is executed because a *righter* r_r is located with a *dumbSearcher* r_d necessarily $id_{r_r} > idPotentialMin_{r_d}$, otherwise it is not possible for r_r to execute Rule $\mathbf{M_7}$, since it would have executed Rule $\mathbf{M_1}$ at the same round (since the predicate $MinDiscovery()$ is true because $(kind_{r_d} \in \{dumbSearcher, potentialMin\} \wedge id_{r_r} < idPotentialMin_{r_d})$). Therefore if Rule $\mathbf{M_7}$ is executed at round t because a *righter* r_r is located with a *dumbSearcher* r_d, this implies, by the predicate $DumbSearcherMinRevelation()$ of Rule $\mathbf{M_9}$, that Rule $\mathbf{M_9}$ is also executed at round t. Indeed, r_r executes Rule $\mathbf{M_7}$, while r_d executes Rule $\mathbf{M_9}$. The reverse is also true: if a *dumbSearcher* r_d executes Rule $\mathbf{M_9}$ at round t, then necessarily a *righter* r_r, such that $id_{r_r} > idPotentialMin_{r_d}$, executes Rule $\mathbf{M_7}$ at round t. While executing respectively these rules the two robots update their variables $idMin$ with the value of the variable $idPotentialMin$ of the *dumbSearcher*. By using the same arguments as the one used in case 1, we know that $idPotentialMin$ is the identifier of r_{min}. Therefore the variables $idMin$ are either set with the identifier of r_{min} while Rules $\mathbf{M_7}$ and $\mathbf{M_9}$ are executed, or copied from another robots while Rules $\mathbf{M_5}$ or $\mathbf{M_{10}}$ are executed. However whatever the rule executed the value of $idMin$ is set with the identifier of r_{min}.

Case 3: r becomes min because the condition "$\exists \mathbf{r'} \in \mathbf{NodeMate}()$, $(\mathbf{kind_{r'}} \in \{\mathbf{dumbSearcher}, \mathbf{potentialMin}\} \wedge \mathbf{id_r} < \mathbf{idPotentialMin_{r'}})$" is true.

Only the execution of Rule $\mathbf{M_6}$ permits to have *dumbSearcher* or $potentialMin$ in the execution. This rule is executed when $\mathcal{R}-1$ *righter* robots are on a same node. When executing this rule, the $\mathcal{R}-1$ robots set their variables $idPotentialMin$ to the identifier of the robot that possesses the minimum identifier among them. Moreover among the $\mathcal{R}-1$ robots that execute Rule $\mathbf{M_6}$, one robot becomes $potentialMin$ while the other become *dumbSearcher*. Besides if a robot is a *dumbSearcher* (resp. a $potentialMin$) in a configuration γ_t at time t and is still a *dumbSearcher* (resp. a $potentialMin$) in the configuration γ_{t+1} then it does not update its variable $idPotentialMin$ during time t since it executes Rule $\mathbf{M_{11}}$ (resp. $\mathbf{M_8}$). As Rule $\mathbf{M_6}$ can only be executed once (see the arguments of case 1), if r meets a *dumbSearcher* or a $potentialMin$ r', such that $id_r < idPotentialMin_{r'}$, this necessarily implies that r' is issued of the execution of Rule $\mathbf{M_6}$ while r has not executed this rule, and therefore $r = r_{min}$, which is a contradiction.

Case 4: r becomes min because $\mathbf{rightSteps_r = 4 * id_r * n}$.

At the time where r becomes min, r_{min} is either a *righter* robot, a $potentialMin$ robot or min, otherwise this implies that there already exists a min (other than r_{min}) in the execution, which is a contradiction with the fact that r is the first robot different from r_{min} that becomes min.

By the predicate $PotentialMinOrRighter()$ of Rule $\mathbf{M_1}$, only *righter* robots or $potentialMin$ robots can become min. By Lemma 6.2, if, at a time t, a robot is a *righter* or a $potentialMin$, then it points to the *right* direction from the beginning of the execution until the Look phase of time t. Robots that are *righter* robots or $potentialMin$ robots in a

configuration γ_t at time t and that are either *righter* or *potentialMin* in the configuration γ_{t+1} increase from 1 their variables *rightSteps* each time an adjacent edge in the right direction to their positions is present (Rules $\mathbf{M_6}$ and $\mathbf{M_8}$). Therefore, by the predicate $MinDiscovery()$ of Rule $\mathbf{M_1}$ a robot r" moves at most during $4 * id_{r"} * n$ steps in the right direction before being *min*.

By Lemma 6.1, from the time a robot becomes *min*, it is either a $minWaitingWalker$ or a $minTailWalker$. Therefore it can only execute Rules $\mathbf{Term_1}$, $\mathbf{Term_2}$, $\mathbf{K_1}$, $\mathbf{K_2}$, $\mathbf{W_1}$ and $\mathbf{T_3}$. This implies that once a robot is *min*, it only points either to the *right* or to the $\perp$ direction, and can move during at most n steps in the right direction before stopping to move definitively (by executing the following rules in the order: $\mathbf{K_2}$, $\mathbf{K_1}$, $\mathbf{W_1}$ and $\mathbf{T_3}$). Therefore by the previous paragraph, a *min* r" points to the right or the $\perp$ direction from the beginning of the execution until the end of the execution, and can move during at most $4 * id_{r"} * n + n$ steps in the right direction during the whole execution.

Because of the dynamism of the ring, by Observation 6.1 and since when a *righter* or a *potentialMin* robot stops to be a *righter* or a *potentialMin* robot, it stops to update the value of its variable *rightSteps*, we have: $\forall r_1, r_2 \in \mathcal{R}^2, kind_{r_1}, kind_{r_2} \in \{righter, potentialMin\}^2, |rightSteps_{r_1} - rightSteps_{r_2}| \leq n$.

Because it takes one round for a robot to update its variable *kind* to *min*, a *righter* or a *potentialMin* can be located with a robot r just the round before r becomes *min*. Therefore this *righter* or *potentialMin* can move again in the right direction during at most n steps without meeting the *min*.

We know that $id_{r_{min}} < id_r$, therefore we have $4 * id_{r_{min}} * n + n + n + n < 4 * id_r * n$. Hence there exists a time at which r meets r_{min} while r_{min} is *min* and r is not yet *min*. At this time, by the rules of $\mathbb{GDG}$, r stops being a *righter* or a *potentialMin* robot, and hence by Observation 6.1, r cannot be anymore a *righter* robot or a *potentialMin* robot and therefore it cannot become *min*, which leads to a contradiction.

□

The following lemma helps us to prove the Lemma 6.5. This lemma is true only if there is no *min* in the execution. In other words, it is true only if all the robots are executing Phase `M`.

Lemma 6.4. *If there is no min in the execution, if, at time t, a robot r is such that* $kind_r \in \{dumbSearcher, awareSearcher\}$*, then, during the Move phase of time* $t - 1$*, it does not point to the* $\perp$ *direction.*

Proof. Consider a robot r such that, at time t, $kind_r \in \{dumbSearcher, awareSearcher\}$.

While executing $\mathbb{GDG}$, since initially all the robots are *righter*, if there is no *min*, only *righter*, *potentialMin*, *dumbSearcher* and *awareSearcher* robots can be present in the execution.

Consider then the two following cases.

Case 1: At time t − 1, r is neither a *dumbSearcher* nor an *awareSearcher*.

Whatever the value of the variable *kind* of r at time $t - 1$ (*righter* or *potentialMin*), to have its variable *kind* at time t equals either to *dumbSearcher* or to *awareSearcher*, r executes at time $t - 1$ either Rule $\mathbf{M_5}$, $\mathbf{M_6}$ or $\mathbf{M_7}$.

Consider first the case where r executes Rule $\mathbf{M_6}$ at time $t - 1$. Only *righter* robots can execute Rule $\mathbf{M_6}$. While executing Rule $\mathbf{M_6}$, r becomes a *dumbSearcher* (since while executing this rule a *righter* can become either a *dumbSearcher* or a *potentialMin*). Moreover, while executing Rule $\mathbf{M_6}$, a *righter* that becomes *dumbSearcher* does not change the direction it points to. By Lemma 6.2, during the Look phase of time $t - 1$, r points to the

right direction and since r does not change its direction during the Compute phase of time $t-1$, this implies that the lemma is proved in this case.

Consider now the case where r executes either Rule $\mathbf{M_5}$ or $\mathbf{M_7}$. While executing these rules the function SEARCH() is called.

While executing the function SEARCH(), if there are multiple robots on the current node of r at time $t-1$, it points either to the right or to the left direction. Therefore, in this case the lemma is proved.

In the case r is alone on its node at time $t-1$, while executing the function SEARCH() it does not change its direction. Moreover, while executing Rules $\mathbf{M_5}$ or $\mathbf{M_7}$, before calling the function SEARCH() the robot calls the function BECOMEAWARESEARCHER(_) that sets its direction to the right direction. Therefore, in these cases, even if r is alone on its node, it points to a direction different from $\perp$ during the Move phase of time $t-1$, hence the lemma is proved.

Case 2: At time t − 1, r is a *dumbSearcher* or an *awareSearcher*.

Whatever the value of the variable *kind* of r at time $t-1$ (*dumbSearcher* or *awareSearcher*), to have its variable *kind* at time t equals either to *dumbSearcher* or to *awareSearcher*, r executes at time $t-1$ either Rule $\mathbf{M_9}$, $\mathbf{M_{10}}$ or $\mathbf{M_{11}}$. While executing these rules the function SEARCH() is called.

As highlighted in the case 1, if there are multiple robots on the current node of r at time $t-1$, the lemma is proved.

Moreover, while executing Rules $\mathbf{M_9}$ and $\mathbf{M_{10}}$, before calling the function SEARCH() the robot calls the function BECOMEAWARESEARCHER(_) that sets its direction to the right direction. Therefore, in these cases, even if r is alone on its node, it points to a direction different from $\perp$ during the Move phase of time $t-1$, hence the lemma is proved.

It remains the case where r executes Rule $\mathbf{M_{11}}$ at time $t-1$ while it is alone on its node. In this case, while executing Rule $\mathbf{M_{11}}$, r does not change its direction (refer to the function SEARCH()). Since at time $t-1$, r is already a *dumbSearcher* or an *awareSearcher*, and since initially all the robots are *righter*, by recurrence on all the cases treated previously (Case 1 and 2), the direction r points to during the Move phase of time $t-1$ cannot be equal to $\perp$.

□

Finally, we prove the other main lemma of this phase: we prove that r_{min} is aware, in finite time, that it possesses the minimum identifier among all the robots of the system.

Lemma 6.5. *In finite time r_{min} becomes min.*

Proof. Assume that r_{min} does not become *min*. By Lemma 6.3, only r_{min} can be *min*. While executing $\mathbb{GDG}$, since initially all the robots are *righter*, if there is no *min*, only *righter*, *potentialMin*, *dumbSearcher* and *awareSearcher* robots can be present in the execution.

Initially all the robots are *righter*. In the case where there is no *min* in the execution, by the rules of $\mathbb{GDG}$, from a configuration γ_t at a time t where there are only *righter* robots, it is not possible to have *awareSearcher* in the configuration γ_{t+1}. A robot can become a *dumbSearcher* or a *potentialMin* only when Rule $\mathbf{M_6}$ is executed. This rule is executed when $\mathcal{R}-1$ *righter* robots are on a same node (refer to predicate *AllButOneRighter*()).

Let us now consider the three following cases that can occur in the execution.

Case 1: Rule $\mathbf{M_6}$ is never executed.

In this case all the robots are *righter* robots during the whole execution, and execute therefore Rule $\mathbf{M_8}$ at each instant time. While executing Rule $\mathbf{M_8}$, a robot always points to the *right* direction and increments its variable *rightSteps* by one each time there exists an adjacent *right* edge to its location. Since by assumption r_{min} does not become *min*, then by Rule $\mathbf{M_1}$ and predicate $MinDiscovery()$, r_{min} cannot succeed to have its variable *rightSteps* equals to $4 * id_{r_{min}} * n$, otherwise the lemma is true. Therefore it exists a time at which r_{min} is on a node such that its adjacent *right* edge is missing forever. Since it can exist at most one eventual missing edge in a $\mathcal{COT}$ ring, and since all the robots always move in the *right* direction when there is an adjacent right edge to their location (since they execute Rule $\mathbf{M_8}$), it exists a time at which $\mathcal{R}-1$ *righter* robots are on a same node, cases 2 and 3 are then considered.

Case 2: Rule $\mathbf{M_6}$ is executed but $\mathbf{r_{min}}$ is not among the $\mathcal{R}-1$ righter robots that execute it.

While executing Rule $\mathbf{M_6}$, among the $\mathcal{R}-1$ *righter* located on a same node that execute this rule, the robot with the minimum identifier r_p becomes $potentialMin$ while the other robots become *dumbSearcher*, and all update their variables $idPotentialMin$ to id_{r_p}. By definition we have $id_{r_p} > id_{r_{min}}$. By Observation 6.1, a robot that becomes a *dumbSearcher* can never become *righter* robot or $potentialMin$ robot. Moreover, by Observation 6.2, a robot that becomes a $potentialMin$ can never become a *righter*. Since $\mathcal{R}-1$ *righter* are needed to execute Rule $\mathbf{M_6}$, this rule can be executed only once. Note that if a robot is a *dumbSearcher* (resp. a $potentialMin$) in a configuration γ_t at time t and is still a *dumbSearcher* (resp. a $potentialMin$) in the configuration γ_{t+1} then it does not update its variable $idPotentialMin$ during time t since it executes Rule $\mathbf{M_{11}}$ (resp. $\mathbf{M_8}$)

At the time of the execution of Rule $\mathbf{M_6}$, r_{min} is a *righter*, since it is not among the robots that execute this rule. After the execution of this rule r_{min}, as a *righter*, cannot meet a $potentialMin$ robot. Indeed the only way for a robot to become $potentialMin$ is to execute Rule $\mathbf{M_6}$. Therefore only r_p can be $potentialMin$, and we know that $idPotentialMin_{r_p} = id_{r_p} > id_{r_{min}}$. Hence if r_{min} meets a $potentialMin$, then by Rule $\mathbf{M_1}$ and predicate $MinDiscovery()$ the lemma is true, which is a contradiction.

Similarly, r_{min} as a *righter* cannot meet a *dumbSearcher* r_d. Indeed, only Rule $\mathbf{M_6}$ permits a robot to become a *dumbSearcher*. Therefore, since $idPotenialMin_{r_d} = id_{r_p} > id_{r_{min}}$, if r_{min} meets a *dumbSearcher*, then by Rule $\mathbf{M_1}$ and predicate $MinDiscovery()$ the lemma is true, which is a contradiction.

Moreover it cannot exist *awareSearcher* in this execution. Indeed, as said previously, from a configuration γ_t at a time t where there are only *righter* robots, it is not possible to have *awareSearcher* in the configuration γ_{t+1}. Therefore *awareSearcher* can be present in the execution only after the execution of Rule $\mathbf{M_6}$. In the case where there is not yet *awareSearcher*, a robot can become an *awareSearcher* only if a *righter* meets a *dumbSearcher* (Rules $\mathbf{M_9}$ and $\mathbf{M_7}$). However after the execution of Rule $\mathbf{M_6}$, only r_{min} is a *righter*, and as explained in the previous paragraph, if r_{min} as a *righter* meets a *dumbSearcher* there is a contradiction.

Since there is no *awareSearcher* and since r_{min} as a *righter* can meet neither $potentialMin$ nor *dumbSearcher*, this implies that r_{min} stays a *righter* during the whole execution and therefore executes Rule $\mathbf{M_8}$ at each instant time. By the same arguments as the one used in case 1, necessarily it exists a time at which r_{min} is on node such that its adjacent *right* edge is missing forever, otherwise the lemma is true. However since there is no *min* is the execution, and there is no *awareSearcher*, r_p stays a $potentialMin$ and executes Rule $\mathbf{M_8}$ at each instant time, therefore it always points to the *right* direction. Since it can only exist one eventual missing edge and since this edge is the adjacent *right* edge to the

position of r_{min}, all the other edges are infinitely often present. Therefore, in finite time, the $potentialMin$ is located on the same node as r_{min}, which is a contradiction.

Case 3: Rule $\mathbf{M_6}$ is executed and $\mathbf{r_{min}}$ is among the $\mathcal{R}-1$ righter robots that execute it.

We use the same arguments as the one used in case 2. Therefore we know that while executing Rule $\mathbf{M_6}$, r_{min} becomes $potentialMin$, since r_{min} possesses the minimum identifier among all the robots of the system.

Moreover, since r_{min} does not become min, as a $potentialMin$, it cannot meet a $righter$ robot otherwise by Rule $\mathbf{M_1}$ and predicate $MinDiscovery()$ the lemma is true.

Similarly, r_{min} as a $potentialMin$ cannot meet $awareSearcher$. Indeed in the case there is not yet $awareSearcher$, a robot can become an $awareSearcher$ only if a $righter$ meets a $dumbSearcher$ (Rules $\mathbf{M_9}$ and $\mathbf{M_7}$). While executing these rules a robot that becomes an $awareSearcher$ sets its variable $idMin$ to the identifier of the variable $potentialMin$ of the $dumbSearcher$, which is in this case $id_{r_{min}}$. An $awareSearcher$ never updates the value of its variable $idMin$. Once there is at least one $awareSearcher$ in the execution, it is possible to have other robots that become $awareSearcher$ thanks to the execution of Rule $\mathbf{M_{10}}$. However while executing this rule, a robot that becomes $awareSearcher$ copies the value of the variable $idMin$ of the $awareSearcher$ it is located with. Therefore if r_{min}, as a $potentialMin$, meets an $awareSearcher$, by Rule $\mathbf{M_1}$ and predicate $MinDiscovery()$, the lemma is true, which is a contradiction.

Therefore, as a $potentialMin$, r_{min} executes Rule $\mathbf{M_8}$ at each instant time. By the same arguments as the one used in case 1, necessarily it exists a time at which r_{min} is on node such that its adjacent $right$ edge is missing forever, otherwise the lemma is true.

By Observation 6.1, $dumbSearcher$ and $awareSearcher$ robots cannot become $righter$ or $potentialMin$. As explained, if there is no meeting between a $dumbSearcher$ robot and a $righter$ robot, it cannot exist $awareSearcher$ robots in the execution. As seen previously, no $righter$ robot can meet r_{min}. At the time where Rule $\mathbf{M_6}$ is executed there is a $righter$ robot r in the execution. In the case r never meets a $dumbSearcher$ robot, it executes Rule $\mathbf{M_8}$ at each instant time. Hence, using the arguments as the one used in case 2, in finite time, r can be located on the same node as r_{min}, which is a contradiction. This implies that there exists a time at which r, as a $righter$ robot, meets at least a $dumbSearcher$ robot r'. In this case r executes Rule $\mathbf{M_7}$ (refer to the predicate $RighterWithSearcher()$) and all the $dumbSearcher$ robots located with r including r' execute Rule $\mathbf{M_9}$ (by the predicate $DumbSearcherMinRevelation()$ and since $id_r > id_{r_{min}}$). Hence r and all the $dumbSearcher$ robots located with r become $awareSearcher$ robots and execute the function SEARCH(). When a robot executes the function SEARCH() while there are multiple robots on its node, if it possesses the maximum identifier among the robots of its node, it points to the left direction, otherwise it points to the right direction. Therefore, once $\mathbf{M_7}$ and $\mathbf{M_9}$ are executed, there are at least two $awareSearcher$ considering two opposite directions. Moreover once $\mathbf{M_7}$ and $\mathbf{M_9}$ are executed, except r_{min} there are only $dumbSearcher$ and $awareSearcher$ robots in the execution. When a $dumbSearcher$ robot meets an $awareSearcher$ robot, it executes Rule $\mathbf{M_{10}}$ and therefore becomes $awareSearcher$ robot and executes the function SEARCH(). An $awareSearcher$ executes Rule $\mathbf{M_{11}}$ at each instant time, therefore it calls the function SEARCH() at each instant time. While executing the function SEARCH(), if an $awareSearcher$ robot is alone on its node, it points to the last direction it points to (this direction cannot be equal to $\perp$ by Lemma 6.4). All this implies that in finite time an $awareSearcher$ robot is located on the same node as r_{min}. Therefore by Rule $\mathbf{M_1}$ and predicate $MinDiscovery()$, r_{min} becomes min.

□

By Lemmas 6.3 and 6.5, we can deduce the following corollary which proves the correctness of Phase M.

Corollary 6.4. *r_{min} becomes min in finite time and the other robots never become min.*

Correctness Proof of Phase K

Once r_{min} completes Phase M, it stops to move and waits for the completion of Phase K. We recall that, during Phase K of $\mathbb{GDG}$, $\mathcal{R}-3$ robots must join r_{min} on the node where it is waiting. More precisely, while executing $\mathbb{GDG}$, Phase K is achieved when $\mathcal{R}-3$ $waitingWalker$ robots are located on the node where r_{min}, as min, is waiting. In the previous section, we prove that, in finite time, only r_{min} becomes min (Corollary 6.4) and that once r_{min} is min it stays min for the rest of the execution (Lemma 6.1). Note that, by the rules of $\mathbb{GDG}$, the min is necessarily a $minWaitingWalker$ robot before being a $minTailWalker$ (since only a $minWaitingWalker$ can become a $minTailWalker$ while executing Rule $\mathbf{K_1}$). Moreover, by Rule $\mathbf{K_2}$, r_{min}, as a $minWaitingWalker$, does not move until $\mathcal{R}-3$ $waitingWalker$ robots are on its node. Therefore, as $minWaitingWalker$, r_{min} is, as expected, always on the same node. Let u be the node on which r_{min}, as a $minWaitingWalker$, is located. Let t_{min} be the time at which r_{min} becomes a $minWaitingWalker$ robot. In this section, we consider the execution from time t_{min}.

To simplify the proofs, we introduce the notion of $towerMinConfiguration$ as follows.

Definition 6.1 ($towerMinConfiguration$)**.** *A $towerMinConfiguration$ corresponds to a configuration of the execution in which $\mathcal{R}-3$ $waitingWalker$ robots are located on the same node as the $minWaitingWalker$.*

To prove the correctness of Phase K, we hence have to prove that, in finite time, a $towerMinConfiguration$ is formed.

As noted previously, by the rules of $\mathbb{GDG}$, as long as there is no $towerMinConfiguration$, r_{min} stays a $minWaitingWalker$ robot.

The following observation is useful to prove the correctness of this phase.

Observation 6.3. *There exists no rule in $\mathbb{GDG}$ permitting a robot that stops being either $minWaitingWalker$ or $waitingWalker$ robot to be again a $minWaitingWalker$ or $waitingWalker$ robot.*

To prove the correctness of this phase, we prove, first, that if a $potentialMin$ is present in the execution then, in finite time, a $towerMinConfiguration$ is present in the execution, next, we prove that if there is no $potentialMin$ in the execution then, in finite time, a $towerMinConfiguration$ is also present in the execution. We prove this respectively in Lemmas 6.15 and 6.16. To simplify the proofs of these two lemmas, we need to prove the nine following lemmas.

In the following lemma, we prove that it can exist at most one $towerMinConfiguration$ in the whole execution.

Lemma 6.6. *It can exist at most one $towerMinConfiguration$ in the whole execution.*

Proof. By definition a $towerMinConfiguration$ is composed of one $minWaitingWalker$ and $\mathcal{R}-3$ $waitingWalker$ robots. Once a $towerMinConfiguration$ is formed, the $\mathcal{R}-2$ ($\mathcal{R}-2\geq 2$) robots involved in the $towerMinConfiguration$ execute Rule $\mathbf{K_1}$. While executing this rule the robot with the maximum identifier among the $\mathcal{R}-2$ robots involved in the $towerMinConfiguration$ becomes $headWalker$ while the $minWaitingWalker$ becomes $minTailWalker$ and the other robots involved in the $towerMinConfiguration$ become $tailWalker$.

Then by Observation 6.3 and since by Corollary 6.4 only r_{min} can be $minWaitingWalker$, the lemma is proved. □

In the following lemma, we prove that all the $waitingWalker$ as well as the $minWaiting-Walker$ are located on node u and do not move. This is important to prove that, in finite time, a $towerMinConfiguration$ is formed.

Lemma 6.7. *All $waitingWalker$ robots are located on the same node as r_{min} when $kind_{r_{min}} = minWaitingWalker$ and neither the $waitingWalker$ robots nor r_{min}, as a $minWaitingWalker$, move.*

Proof. By the rules of $\mathbb{GDG}$, as long as there is no $towerMinConfiguration$, r_{min} is $min-WaitingWalker$. While r_{min} is the $minWaitingWalker$, it executes Rule $\mathbf{K_2}$ at each instant time. While executing this rule, r_{min} points to the $\perp$ direction and therefore does not move.

Only Rule $\mathbf{K_3}$ permits a robot r to become a $waitingWalker$ robot. For this rule to be executed r must be located with a $minWaitingWalker$ (refer to predicate $PotentialMinOr-SearcherWithMin()$). By Corollary 6.4, only r_{min} can be $minWaitingWalker$. While executing Rule $\mathbf{K_3}$, r points to the $\perp$ direction and therefore at the time of the execution of this rule, r does not move and is on the node where r_{min}, as a $minWaitingWalker$, is located.

While r is a $waitingWalker$ robot, as long as there is no $towerMinConfiguration$ in the execution, it executes Rule $\mathbf{K_2}$ at each instant time. Therefore r does not move. As noted previously, the location where r stops moving is the location where r_{min}, as the $minWaiting-Walker$, is located.

Once a $towerMinConfiguration$ is present in the execution the $waitingWalker$ robots and the $minWaitingWalker$ composing this $towerMinConfiguration$ execute Rule $\mathbf{K_1}$. While executing this rule the robots do not change the direction they point to and stop being $waiting-Walker/minWaitingWalker$ robots. Therefore, by Observation 6.3 and since by Corollary 6.4 only r_{min} can be $minWaitingWalker$, all $waitingWalker$ robots are located on the same node as r_{min} when $kind_{r_{min}} = minWaitingWalker$ and neither the $waitingWalker$ robots nor r_{min}, as a $minWaitingWalker$, move. □

Now we prove a property on $potentialMin$.

Lemma 6.8. *It can exist at most one $potentialMin$ robot in the whole execution.*

Proof. Only the execution of Rule $\mathbf{M_6}$ permits a robot to become a $potentialMin$ robot. Rule $\mathbf{M_6}$ is executed when $\mathcal{R} - 1$ $righter$ robots are located on a same node. When these $\mathcal{R} - 1$ $righter$ robots execute Rule $\mathbf{M_6}$, one becomes a $potentialMin$, and the others become $dumb-Searcher$. Therefore, by Observations 6.1 and 6.2 this rule can be executed only once. Moreover, by the rules of $\mathbb{GDG}$, once a $potentialMin$ stops to be a $potentialMin$, it cannot be again a $potentialMin$. Hence the lemma is proved. □

The following lemma demonstrates a property on min.

Lemma 6.9. *Before being min, r_{min} is either a $righter$ robot or a $potentialMin$ robot.*

Proof. A robot that is a min is a robot such that its variable $kind$ is either equal to $minWaiting-Walker$ or to $minTailWalker$. The only way to be a $minTailWalker$ robot is to be a $min-WaitingWalker$ robot and to execute Rule $\mathbf{K_1}$. The only way to be a $minWaitingWalker$ is to execute Rule $\mathbf{M_1}$. Only $righter$ robots or $potentialMin$ robots can execute Rule $\mathbf{M_1}$ (refer to predicate $PotentialMinOrRighter()$). □

The three following lemmas give properties on the execution, when r_{min} is min. Indeed, they indicate the presence or absence of $righter/potentialMin$ in the execution while r_{min} is min.

Lemma 6.10. *In the suffix of the execution starting from the time where r_{min} is min, it is not possible to have a $potentialMin$ robot and a $righter$ robot present at the same time.*

Proof. By Lemma 6.9, r_{min} is either a *righter* or a *potentialMin* before being *min*. In the case where r_{min} is a *potentialMin* before being *min*, then by Lemma 6.8, it cannot exist a *potentialMin* in the execution after r_{min} becomes *min*. Therefore the lemma is proved in this case.

Consider now the case where r_{min} is a *righter* before being *min*. For a robot to become a *potentialMin* Rule $\mathbf{M_6}$ must be executed. This rule is executed when $\mathcal{R}-1$ *righter* are located on a same node. While executing Rule $\mathbf{M_6}$, among the $\mathcal{R}-1$ *righter* located on a same node, the one with the minimum identifier becomes *potentialMin* while the others become *dumbSearcher*. By Observation 6.2, r_{min} cannot be among the $\mathcal{R}-1$ *righter* that execute Rule $\mathbf{M_6}$, otherwise it cannot be a *righter* before being *min*. Similarly thanks to Observation 6.2, the $\mathcal{R}-1$ robots that execute Rule $\mathbf{M_6}$, cannot be *righter* anymore after the execution of this rule. therefore, it is not possible to have a *potentialMin* and a *righter* in the execution once r_{min} is *min*. □

Lemma 6.11. *If there exists a time t at which a righter, a robot r ($r \neq r_{min}$) such that $kind_r \neq$ righter and r_{min}, as min, are present in the execution, then there is no more potentialMin in the suffix of the execution starting from t.*

Proof. By Lemma 6.10, since there is a *righter* at time t, there is no *potentialMin* in the execution at time t.

Since at time t, r_{min} and r are not *righter* and can never be *righter* anymore (refer to Observation 6.2), it is not possible to have $\mathcal{R}-1$ *righter* located on a same node after time t. However, in order to have a *potentialMin* in the execution, Rule $\mathbf{M_6}$ must be executed. This rule is executed only if $\mathcal{R}-1$ *righter* are located on a same node. Therefore there is no *potentialMin* in the execution after time t. □

Lemma 6.12. *If there is a potentialMin at a time t, and if before being min, r_{min} is a righter, then there is no more righter in the suffix of the execution starting from time $t' = max\{t, t_{min}\}$.*

Proof. Assume that before being *min*, r_{min} is a *righter*. Moreover assume that there is a *potentialMin* in the execution at time t.

($*$) For a robot to become a *potentialMin* Rule $\mathbf{M_6}$ must be executed. This rule is executed when $\mathcal{R}-1$ *righter* are located on a same node. While executing Rule $\mathbf{M_6}$, among the $\mathcal{R}-1$ *righter* located on a same node, the one with the minimum identifier becomes *potentialMin* while the others become *dumbSearcher*. By Observation 6.2 none of these $\mathcal{R}-1$ robots can become *righter* anymore after time t.

Consider then the two following cases.

Case 1: $\mathbf{t > t_{min}}$.

By Observation 6.2, r_{min} cannot be a *righter* after time t_{min}. Therefore r_{min} is not among the $\mathcal{R}-1$ robots that execute Rule $\mathbf{M_6}$, and hence, by ($*$), after time t, there is no more *righter* in the execution.

Case 2: $\mathbf{t \leq t_{min}}$.

By ($*$), r_{min} cannot be among the $\mathcal{R}-1$ *righter* that execute Rule $\mathbf{M_6}$, otherwise it cannot be a *righter* before being *min*. Therefore, by ($*$) and since after time t_{min}, by Observation 6.2, r_{min} cannot be a *righter* anymore, there is no more *righter* in the execution after time t_{min}.

□

The following lemma is an extension of Lemma 6.4. While Lemma 6.4 is true only when all the robots are executing Phase `M`, the following lemma is true whether the robots are executing Phase `M` or Phase `K`.

Lemma 6.13. *If there is no $towerMinConfiguration$ in the execution, if, at time t, a robot r is such that $kind_r \in \{potentialMin, dumbSearcher, awareSearcher\}$, then, during the Move phase of time $t-1$, it does not point to the $\perp$ direction.*

Proof. Consider a robot r such that at time t, $kind_r = potentialMin$. By Lemma 6.2, r points to the right direction during the Move phase of time $t-1$. Hence the lemma is proved in this case.

Consider now a robot r such that, at time t, $kind_r \in \{dumbSearcher, awareSearcher\}$. Since there is no $towerMinConfiguration$ in the execution, by the rules of $\mathbb{GDG}$ and knowing that initially all the robots are *righter*, there are only *righter*, $potentialMin$, *dumbSearcher*, *awareSearcher*, $waitingWalker$ and $minWaitingWalker$ robots in the execution. Note that there is no rule in $\mathbb{GDG}$ permitting a $waitingWalker$ or a $minWaitingWalker$ to become either a *dumbSearcher* or an *awareSearcher*.

Consider then the two following cases.

Case 1: At time t − 1, r is neither a *dumbSearcher* nor an *awareSearcher*.

Whatever the value of the variable *kind* of r at time $t-1$ (*righter* or $potentialMin$), to have its variable *kind* at time t equals either to *dumbSearcher* or to *awareSearcher*, r executes at time $t-1$ either Rule $\mathbf{K_4}$, $\mathbf{M_5}$, $\mathbf{M_6}$ or $\mathbf{M_7}$.

When a robot executes Rule $\mathbf{K_4}$, it calls the function BecomeAwareSearcher(_). When a robot executes the function BecomeAwareSearcher(_), it sets its direction to the *right* direction, therefore the lemma is also true in this case.

Then, we can use the arguments of the proof of Lemma 6.4 to prove that the current lemma is true for the remaining cases. Indeed, even if in Lemma 6.4 the context is such that there is no *min* in the execution, the arguments used in its proof are still true in the context of the current lemma.

Case 2: At time t − 1, r is a *dumbSearcher* or an *awareSearcher*.

Whatever the value of the variable *kind* of r at time $t-1$ (*dumbSearcher* or *awareSearcher*), to have its variable *kind* at time t equals either to *dumbSearcher* or to *awareSearcher*, r executes at time $t-1$ either Rule $\mathbf{M_9}$, $\mathbf{M_{10}}$ or $\mathbf{M_{11}}$. Similarly as for the case 1, we can use the arguments of the proof of Lemma 6.4 to prove that the current lemma is true in these cases.

□

The following lemma proves that in the case where there are at least 3 robots in the execution such that they are either $potentialMin$, *dumbSearcher* or *awareSearcher*, then, in finite time, at least one of these kinds of robots is located on node u. A $potentialMin$, a *dumbSearcher* or an *awareSearcher* located with the $minWaitingWalker$ becomes a $waitingWalker$ (Rule $\mathbf{K_3}$). Therefore, this lemma permits to prove that in the case where there are at least 3 robots in the execution (after time t_{min}) such that they are either $potentialMin$, *dumbSearcher* or *awareSearcher*, then, in finite time, a supplementary $waitingWalker$ is located on node u.

To prove the following lemma, we need to use the $Seg(u, v)$ notion (refer to Section 3.2.1). We recall that $Seg(u, v)$ is the set of nodes (in the footprint of the dynamic ring) between node u not included and v not included considering the right direction of the robots of the system.

Lemma 6.14. *If there is no $towerMinConfiguration$ in the execution but there exists at a time t at least 3 robots such that they are either $potentialMin$, dumbSearcher or awareSearcher, then it exists a time $t' \geq t$ at which at least a $potentialMin$, a dumbSearcher or an awareSearcher, reaches the node u.*

Proof. Assume that there is no $towerMinConfiguration$ in the execution. By the rules of $\mathbb{GDG}$ and knowing that initially all the robots are $righter$, this implies that there are only $righter$, $potentialMin$, $dumbSearcher$, $awareSearcher$, $waitingWalker$ and $minWaitingWalker$ robots in the execution. Since there is no $towerMinConfiguration$, r_{min} is $minWaitingWalker$ and is located on node u. By Lemma 6.7, we know that all the $waitingWalker$ robots (if any) are on node u and r_{min} as well as the $waitingWalker$ robots do not move. This implies that among the robots that are not on node u there are only $righter$, $potentialMin$, $dumbSearcher$ and $awareSearcher$.

Assume by contradiction that at a time t, there are at least 3 robots such that they are either $potentialMin$, $dumbSearcher$ or $awareSearcher$ and such that for all time $t' \geq t$ none of these kinds of robots succeed to reach the node u at time t'. We consider the execution from time t.

Consider a robot r such that at time t, $kind_r \in \{potentialMin, dumbSearcher, awareSearcher\}$.

(i) If r is an $awareSearcher$, since it cannot reach u, it executes Rule $\mathbf{M_{11}}$, and hence it executes the function SEARCH(). The variable $kind$ of r is not updated while r executes this function, therefore r is an $awareSearcher$ and executes Rule $\mathbf{M_{11}}$ and the function SEARCH() at each instant time from time t. Thus by Lemma 6.13, r always points to a direction different from $\perp$ after time t included.

(ii) If r is a $dumbSearcher$, since it cannot reach u, it can execute either Rule $\mathbf{M_{10}}$ (if it is on the same node as an $awareSearcher$) and hence becomes an $awareSearcher$ robot and executes the function SEARCH(), Rule $\mathbf{M_9}$ (if it is on the same node as a $righter$) and hence becomes an $awareSearcher$ and executes the function SEARCH(), or Rule $\mathbf{M_{11}}$ and hence stays a $dumbSearcher$ and executes the function SEARCH(). By Lemma 6.13 and by (i), r always points to a direction different from $\perp$ after time t included.

(iii) If r is a $potentialMin$, by Lemma 6.10, there is no $righter$ in the execution at time t and therefore by Observation 6.2 there is no $righter$ in the execution after time t included. Therefore, since r cannot reach u, it can execute either Rule $\mathbf{M_5}$ (if it is on the same node as an $awareSearcher$) and hence becomes an $awareSearcher$ and executes the function SEARCH(), or Rule $\mathbf{M_8}$ and hence stays a $potentialMin$ and points to the right direction. Therefore by Lemma 6.13 and by (i), r always points to a direction different from $\perp$ after time t included.

(iv) If there is a $righter$ robot in the execution after time t, then by Lemma 6.11, there is no $potentialMin$ robot in the execution after time t included. If a $righter$ robot is on the same node as a $dumbSearcher$ or as an $awareSearcher$, it executes Rule $\mathbf{M_7}$ and hence becomes an $awareSearcher$ and executes the function SEARCH().

(v) While executing the function SEARCH() if a robot is isolated it points to the last direction it pointed to. While executing the function SEARCH() if a robot is not isolated, if it possesses the maximum identifier among all the robots of its current location it points to the left direction otherwise it points to the right direction.

(vi) Note that if there is a $potentialMin$ while r_{min} is $minWaitingWalker$, then it possesses the minimum identifier among all the robots not located on node u. Indeed, only Rule $\mathbf{M_6}$ permits a robot to become a $potentialMin$. For this rule to be executed, $\mathcal{R} - 1$ $righter$ robots must be located on a same node. While executing this rule the robot with the minimum identifier among the $\mathcal{R} - 1$ robots located on a same node becomes $potentialMin$. Since, by Lemma 6.8, there is only one $potentialMin$ in the whole execution and since by definition r_{min} possesses the minimum identifier among all the robots of the system, r_{min} does not execute Rule $\mathbf{M_6}$. Therefore, while r_{min} is $minWaitingWalker$, the $potentialMin$ possesses the minimum identifier among all the robots not located on node u. Thus, when a $potentialMin$ executes Rule $\mathbf{M_8}$ and hence points to the right direction, it possesses the same behavior as if it was executing the function SEARCH().

Case 1: There is no eventual missing edge.

Call d the direction of r during the Look phase of time t, and let v be the node where r is located during the Look phase of time t. Call w the adjacent node of v in the direction d.

Call e the edge between v and w. As proved in cases (i), (ii) and (iii), d is either equal to $right$ or $left$.

We want to prove that it exists a time t' ($t' \geq t$) such that a robot r' (it is possible to have $r' = r$) with $kind_{r'} \in \{potentialMin, dumbSearcher, awareSearcher\}$ points to the direction d and is located on w during the Look phase of time t'.

Call t" (t" $\geq t$) the first time after time t included where there is an adjacent edge to v. If during the Move phase of time t", r does not point to the direction d, by $(i) - (vi)$ this necessarily implies that when r executes the function SEARCH() (or a function that behaves like the function SEARCH()) there is at least another robot on its node. Moreover by $(i)-(vi)$ the other robot(s) with r also executes the function SEARCH() (or a function that behaves like the function SEARCH()) and is or becomes $potentialMin$, $dumbSearcher$ or $awareSearcher$. Therefore, since all the robots possess distinct identifiers and by (v), during the Move phase of time t", a robot among $\{potentialMin, dumbSearcher, awareSearcher\}$ on node v points to the direction d.

Since all the edges are infinitely often present, we can repeat these arguments on each instant time until the time t_e where e is present. At time t_e a robot (either $potentialMin$, $dumbSearcher$, $awareSearcher$) points to the direction d and hence crosses e. Since the direction pointed to by a robot can be updated only during Compute phases, we succeed to prove that t' exists.

Applying these arguments recurrently we succeed to prove that in finite time a robot r" such that $kind_{r"} \in \{potentialMin, dumbSearcher, awareSearcher\}$ is on node u.

Case 2: There is an eventual missing edge.

Call e the eventual missing edge. Consider the execution after the time greater or equal to t where e is missing forever. Call v the node such that its adjacent right edge is e. Call w the adjacent right node of v.

At least two robots that are either $potentialMin$, $dumbSearcher$ or $awareSearcher$ are either on nodes in $Seg(u, v) \cup \{v\}$ or on nodes in $Seg(w, u) \cup \{w\}$.

Assume that there are at least two robots that are either $potentialMin$, $dumbSearcher$ or $awareSearcher$ which are on nodes in $Seg(u, v) \cup \{v\}$. The reasoning when there are at least two robots that are either $potentialMin$, $dumbSearcher$ or $awareSearcher$ which are on nodes in $Seg(w, u) \cup \{w\}$ is similar.

The edge e is an eventual missing edge. It can exist only one eventual missing edge in $\mathcal{COT}$ ring. Therefore all the edges between the nodes in $\{u\} \cup Seg(u, v) \cup \{v\}$ are infinitely often present. Thus, if there exists a robot (either $potentialMin$, $dumbSearcher$ or $awareSearcher$) that points to the left direction then we can apply the arguments of case 1 to prove that in finite time a robot r", such that $kind_{r"} \in \{potentialMin, dumbSearcher, awareSearcher\}$, is on node u.

Therefore consider that all the robots, that are either $potentialMin$, $dumbSearcher$ or $awareSearcher$ and that are located on nodes in $Seg(u, v) \cup \{v\}$, point to the right direction. In this case a robot either $potentialMin$, $dumbSearcher$ or $awareSearcher$ cannot be located on the same node as a robot either $righter$, $potentialMin$, $dumbSearcher$ or $awareSearcher$, otherwise during the Move phase of the time of this meeting, by $(i) - (vi)$, it exists a robot either $potentialMin$, $dumbSearcher$ or $awareSearcher$ that points to the left direction.

Since e is an eventual missing edge, and since there are at least two robots either $potentialMin$, $dumbSearcher$ or $awareSearcher$ that point to the right direction, applying the arguments of case 1 on two of these robots, we succeed to prove that in finite time two of

these robots are located on v. Therefore, by the previous paragraph, in finite time a robot r", such that $kind_{r"} \in \{potentialMin, dumbSearcher, awareSearcher\}$, is on node u.

□

Now, we prove one of the two main lemmas of this phase: we prove that if a *potentialMin* is present in the execution, then, in finite time, a $towerMinConfiguration$ is present in the execution. While proving this lemma, we also prove that, at the time when the $towerMinConfiguration$ is formed, among the two robots not involved in this $towerMinConfiguration$, it can exit at most one *righter*. This information is useful to prove Phase `T`.

Lemma 6.15. *If there is a potentialMin in the execution, then there exists a time t at which a towerMinConfiguration is present and among the robots not involved in the towerMinConfiguration there is at most one righter robot at time t.*

Proof. Assume that there exists a time t at which a $potentialMin$ robot is present in the execution. Assume by contradiction that there is no $towerMinConfiguration$ in the execution. In the following, we consider the execution from time $t' = max\{t, t_{min}\}$.

Since there is no $towerMinConfiguration$, by the rules of $\mathbb{GDG}$ and knowing that initially all the robots are *righter*, there are in the execution only *righter*, $potentialMin$, *dumbSearcher*, *awareSearcher*, $waitingWalker$ and $minWaitingWalker$ robots. By Lemma 6.7, all the $waitingWalker$ are located on the same node as r_{min}, when $kind_{r_{min}} = minWaitingWalker$, and both r_{min}, as a $minWaitingWalker$, and the $waitingWalker$ robots do not move. By Corollary 6.4, only r_{min} can be a $minWaitingWalker$. We recall that r_{min} as $minWaitingWalker$ is located on node u. Therefore the $minWaitingWalker$ and all the $waitingWalker$ (if any) are located on node u.

By Lemma 6.9 we know that, before being *min*, r_{min} is either a *righter* robot or a $potentialMin$ robot. We can then consider the two following cases.

Case 1: Before being min, $\mathbf{r_{min}}$ is a righter robot.

By Lemma 6.12, at time t' there are only $potentialMin$, *dumbSearcher*, *awareSearcher*, $waitingWalker$ and $minWaitingWalker$ robots in the execution. Moreover, in this case, all the robots that are not on node u are necessarily either $potentialMin$, *dumbSearcher* or *awareSearcher*.

When a $potentialMin$, a *dumbSearcher* or an *awareSearcher* robot meets the $minWaitingWalker$, it executes Rule $\mathbf{K_3}$, hence it becomes a $waitingWalker$ and stops to move.

Then each time there are at least 3 robots in the execution such that they are either $potentialMin$, *dumbSearcher* and/or *awareSearcher*, using Lemma 6.14, we succeed to prove that at least one $potentialMin$, *dumbSearcher* or *awareSearcher* succeeds to join the node u and therefore becomes a $waitingWalker$. Therefore, by Lemma 6.7, a $towerMinConfiguration$ is formed in finite time.

Case 2: Before being min, $\mathbf{r_{min}}$ is a potentialMin.

For a robot to become a $potentialMin$, Rule $\mathbf{M_6}$ must be executed. This rule is executed when $\mathcal{R}-1$ *righter* are located on a same node. While executing this rule, among the $\mathcal{R}-1$ *righter* located on a same node, one becomes $potentialMin$ while the other become *dumbSearcher*. By Observation 6.2, none of these $\mathcal{R}-1$ robots can become *righter* anymore. Therefore, by Lemma 6.8, once r_{min} is *min*, there are only *dumbSearcher*, *awareSearcher*, $waitingWalker$, $minWaitingWalker$ robots and at most one *righter* robot in the execution. Moreover, in this case, among the robots that are not on node u, there are only *dumbSearcher* and *awareSearcher* and at most one *righter*. By the rules of $\mathbb{GDG}$, as

long as a *dumbSearcher* or an *awareSearcher* is not on node u, its variable *kind* stays in $\{dumbSearcher, awareSearcher\}$.

Once r_{min} is min, if there exists a time at which there is no more *righter* robot in the execution, then, using the arguments of case 1, we succeed to prove that a $towerMinConfiguration$ is formed in finite time. Therefore assume that there is always a *righter* robot r in the execution.

When a *dumbSearcher* or an *awareSearcher* robot is located on the same node as the $minWaitingWalker$, it executes Rule $\mathbf{K_3}$, hence it becomes a $waitingWalker$ and stops to move. Then, using multiple times Lemma 6.14 and Lemma 6.7, we know that in finite time there are in the execution only one *righter* and only 2 robots r' and r" such that $kind_{r'}, kind_{r"} \in \{awareSearcher, dumbSearcher\}^2$ (all the other robots are $minWaitingWalker$ and $waitingWalker$ robots and are located on node u). Note that r' (resp. r") cannot be located on node u, otherwise, by Rule $\mathbf{K_3}$, a $towerMinConfiguration$ is formed. Therefore, r' and r" always have their variable *kind* in $\{dumbSearcher, awareSearcher\}$.

When a *righter* robot is located on the same node as an *awareSearcher* or as a *dumbSearcher*, it executes Rule $\mathbf{M_7}$ and becomes an *awareSearcher*. Similarly, if a *righter* is on the same node as a $minWaitingWalker$ while the adjacent right edge to its position is present, then the *righter* robot executes Rule $\mathbf{K_4}$ and becomes an *awareSearcher*. Therefore, as highlighted previously, these situations cannot happen, otherwise a $towerMinConfiguration$ is formed in finite time. This implies that, as long as the robot r is not on node u, it must be isolated. Since r' and r" cannot be located on node u, if r succeeds to join the node u in the case there is no present adjacent right edge to u, then r executes Rule $\mathbf{M_8}$ and therefore stays a *righter* and points to the right direction. Therefore, since an isolated *righter* robot always executes Rule $\mathbf{M_8}$, hence always points to the right direction, this implies that either r is on a node v ($v \neq u$) such that the adjacent right edge of v is an eventual missing edge at least from the time where r is on node v (case 2.1) or r succeeds to reach u but the adjacent right edge of u is an eventual missing edge at least from the time where r is on node u (case 2.2).

($*$) When an *awareSearcher* or a *dumbSearcher* is isolated it executes Rule $\mathbf{M_{11}}$, hence executes the function SEARCH(), therefore it points to the last direction it pointed to. By Lemma 6.13, this direction cannot be equal to $\perp$.

($**$) Since only r' and r" have their variable *kind* in $\{dumbSearcher, awareSearcher\}^2$, and since r' and r" cannot be located on node u and cannot be located with r, if a *dumbSearcher* is located on the same node as an *awareSearcher* or if an *awareSearcher* (resp. a *dumbSearcher*) is located on the same node as another *awareSearcher* (resp. *dumbSearcher*), necessarily this means that r' and r" are located on a same node, and there is no other robot on the same node as them. When a *dumbSearcher* is on the same node as an *awareSearcher* it executes Rule $\mathbf{M_{10}}$, hence it becomes an *awareSearcher* and executes the function SEARCH(). When an *awareSearcher* is on the same node as a *dumbSearcher* it executes Rule $\mathbf{M_{11}}$ and hence executes the function SEARCH(). Since r' and r" have distinct identifiers, when an *awareSearcher* and a *dumbSearcher* are on a same node, they both execute the function SEARCH(), therefore one points to the right direction, while the other one points to the left direction. Similarly, if two *awareSearcher* (resp. *dumbSearcher*) robots are on the same node, they both execute Rule $\mathbf{M_{11}}$ and hence the function SEARCH(), therefore one points to the right direction, while the other one points to the left direction.

Case 2.1: Let w be the adjacent node of v in the right direction. It can exist only one eventual missing edge, which is the adjacent right edge of node v. Therefore, if a robot, in $Seg(u, v)$ or in $Seg(w, u)$, points to a direction d and does not change this

direction, it eventually succeeds to move in this direction. Similarly, if a robot is on node w and always points to the right direction, it eventually succeeds to move in this direction $(***)$.

Firstly, assume that only r' (resp. r") is on a node in $Seg(u, v)$. By $(*)$ and $(***)$, r' (resp. r") cannot point to the right direction, otherwise it reaches r in finite time. Therefore r' (resp. r") points to the left direction. By $(*)$ and $(***)$, in finite time, r' (resp. r") succeeds to reach u, implying that a $towerMinConfiguration$ is formed.

Secondly, assume that r' and r" are on nodes in $Seg(u, v)$. By $(*)$, $(**)$ and $(***)$, they cannot meet otherwise one of them reaches u in finite time. Moreover, if they do not meet, none of them can point to the left direction otherwise, by $(*)$ and $(***)$, they reach u in finite time. Therefore, they cannot meet and must point to the right direction. By $(*)$ and $(***)$, in finite time one robot among r' and r" succeeds to reach r, implying that a $towerMinConfiguration$ is formed.

Thirdly, assume that r and r" are on nodes in $Seg(v, u)$. By $(*)$, $(**)$ and $(***)$, they cannot meet otherwise one of them reaches u in finite time. Moreover, if they do not meet, none of them can point to the right direction otherwise, by $(*)$ and $(***)$, they reach u in finite time. Therefore, they cannot meet and must point to the left direction. However, by $(*)$ and $(***)$, since the adjacent right edge of v is missing forever, in finite time r' and r" reach w, which is a contradiction with the fact that they do not meet.

Case 2.2: Applying the arguments used in the case 2.1, when r' and r" are on nodes in $Seg(v, u)$, to r' and r" when there are on nodes in $Seg(u, u)$, we succeed to prove that in finite time at least one of them reaches node u, making Rule $\mathbf{Term_2}$ true, which leads to a contradiction.

□

Finally, we prove the other main lemma of this phase: we prove that even if there is no *potentialMin* in the execution, then, in finite time, a $towerMinConfiguration$ is present in the execution. While proving this lemma, we also prove that, at the time when the $towerMinConfiguration$ is formed, among the two robots not involved in this $towerMinConfiguration$, it can exit at most one *righter*. This information is useful to prove Phase `T`.

Lemma 6.16. *If there is no potentialMin in the execution, then there exists a time t at which a $towerMinConfiguration$ is present and among the robots not involved in the $towerMinConfiguration$ there is at most one righter robot at time t.*

Proof. Assume, by contradiction, that there is no $towerMinConfiguration$ in the execution. By the rules of $\mathbb{GDG}$ and knowing that initially all the robots are *righter*, this implies that there are only *righter*, *potentialMin*, *dumbSearcher*, *awareSearcher*, $waitingWalker$ and $minWaiting-Walker$ robots in the execution.

Assume that there is no *potentialMin* in the execution. If there is no *potentialMin* in the execution, it cannot exist *dumbSearcher* in the execution. Indeed, the only way for a robot to become *dumbSearcher* is to execute Rule $\mathbf{M_6}$. However, when this rule is executed, a robot becomes *potentialMin*. Therefore, there are in the execution only *righter*, *awareSearcher*, $waitingWalker$ and $minWaitingWalker$ robots.

Before time t_{min}, by the rules of $\mathbb{GDG}$, there are only *righter* in the execution. Indeed, by Corollary 6.4, only r_{min} can be $minWaitingWalker$ and it becomes $minWaitingWalker$ at time t_{min}. Moreover, the only way for a robot to become $waitingWalker$ is to execute Rule $\mathbf{K_3}$. In the case where there is no *potentialMin* in the execution, only an *awareSearcher* located with r_{min}, as a $minWaitingWalker$, can execute this rule. Besides, the only ways for a robot to become an *awareSearcher* is either to be a *righter* and to be located with an *awareSearcher* (refer to Rule

$\mathbf{M_7}$), or to be a *righter* and to be located with r_{min}, as a $minWaitingWalker$, while an adjacent right edge to their location is present (refer to Rule $\mathbf{K_4}$). Since initially all the robots are *righter*, the first *awareSearcher* of the execution can be present only thanks to the execution of Rule $\mathbf{K_4}$.

All this implies that, even after time t_{min}, as long as no *righter* robot is on node u with r_{min}, as a $minWaitingWalker$, while there is a present adjacent right edge to u, it can exist neither *awareSearcher* nor $waitingWalker$ in the execution: there is at most one $minWaiting$-$Walker$ and there are at least $\mathcal{R} - 1$ *righter*. Moreover, this implies that as long as the situation described has not happened, all the *righter* robots only execute Rule $\mathbf{M_8}$, hence always point to the right direction.

Consider the execution just after time t_{min}. In this context, necessarily, in finite time, there exists a *righter* robot r that succeeds to reach u (while r_{min} is $minWaitingWalker$). Indeed, if this is not the case, this implies that there exists an eventual missing edge e. Since all the *righter* robots always point to the right direction and since it can exist at most one eventual missing edge, this implies that $\mathcal{R} - 1$ *righter* robots reach in finite time the same extremity of e. Thus, Rule $\mathbf{M_6}$ is executed, which leads to a contradiction with the fact that there is no $potentialMin$ in the execution.

Similarly, necessarily, in finite time, there exists an adjacent right edge to u while r is on u. Indeed, if this is not the case, this implies that the adjacent right edge of u is an eventual missing edge. Since all the *righter* robots always point to the right direction and since it can exist at most one eventual missing edge, in finite time all the *righter* succeed to be located on node u. This implies that Rule $\mathbf{Term_1}$ is executed, which leads to a contradiction.

Therefore there exists a time t' at which r executes Rule $\mathbf{K_4}$. At this time r becomes an *awareSearcher* robot and points to the right direction. We then consider the execution from time t'.

$(*)$ From this time t', as long as there exists *righter* in the execution, it always exists an *awareSearcher* robot r' pointing to the right direction, such that there is no *righter* robots on $Seg(u, v)$, where v is the node where r' is currently located. This can be proved by analyzing the movements of the different kinds of robots that we describe in $(i) - (vii)$.

(i) By Lemma 6.7, all the $minWaitingWalker$ and $waitingWalker$ (if any) are on a same node (which is the node u) and do not move.

(ii) If an *awareSearcher* is located on node u, therefore if it is located with r_{min}, as a $minWaitingWalker$, it executes Rule $\mathbf{K_3}$ and becomes a $waitingWalker$ robot.

(iii) If an *awareSearcher* is on a node different from the node u, the only rule it can execute is Rule $\mathbf{M_{11}}$, in which the function Search() is called. While executing this function, an isolated *awareSearcher* points to the direction it points to during its last Move phase. By Lemma 6.13, this direction cannot be $\perp$.

(iv) If a *righter* robot is located only with other *righter* robots or if it is located on node u, therefore if it is located with r_{min}, as a $minWaitingWalker$, such that there is no adjacent right edge to u, it executes Rule $\mathbf{M_8}$, hence it stays a *righter* and points to the right direction.

(v) If a *righter* robot is with r_{min}, as a $minWaitingWalker$, such that there is an adjacent right edge to u, then it executes Rule $\mathbf{K_4}$ and hence becomes an *awareSearcher*.

(vi) If a *righter* robot is on a node different from node u with an *awareSearcher*, it executes Rule $\mathbf{M_7}$ and therefore becomes *awareSearcher* and executes the function Search().

(vii) Note that by the movements described in (i) to (vi), if a robot executes the function Search(), then all the robots that are on the same node as it also execute this function. While executing the function Search(), if multiple robots are on the same node, one points to the left direction, while the others point to the right direction.

Applying these movements on r' and recursively on the robots that r' meet that point to the right direction after their meeting with r' and so on, we succeed to prove the property $(*)$.

$(**)$ Note that if there exists a time at which there is no more *righter* in the execution, then by applying (ii), Lemma 6.7 and Lemma 6.14 multiple times we succeed to prove that a *tower*-

$MinConfiguration$ is formed. Therefore at least one robot is always a *righter* during the whole execution. Call $\mathcal{S}_r$ the set of *righter* robots that stay *righter* during the whole execution.

Let us consider the following cases.

Case 1: There does not exist an eventual missing edge.

None of the robots of $\mathcal{S}_r$ can be located on the same node as an *awareSearcher*, otherwise, by (vi), they become *awareSearcher*. Therefore, all the robots of $\mathcal{S}_r$ that are not on node u can only point to the right direction (refer to (iv)). Since all the edges are infinitely often present, for each robot r" of $\mathcal{S}_r$, it exists a time at which r" is on node u. Moreover, once on node u, as long as there is no adjacent right edge to u, r" points to the right direction (refer to (iv)), and therefore stays on node u. Thus, since all the edges are infinitely often present, for each robot r" of $\mathcal{S}_r$, it exists a time at which r" is on node u such that an adjacent right edge to u is present. Therefore, by (v), in finite time, all the robots of $\mathcal{S}_r$ are *awareSearcher* robots. Hence, by $(**)$, the lemma is proved.

Case 2: There exists an eventual missing edge.

Call x the node such that its adjacent right edge is the eventual missing edge. Consider the execution after time t' such that the eventual missing edge is missing forever.

Case 2.1: $\mathbf{x = u}$.

None of the robots of $\mathcal{S}_r$ can be located on the same node as an *awareSearcher*, otherwise, by (vi), they become *awareSearcher*. Therefore, all the robots of $\mathcal{S}_r$ that are not on node u can only point to the right direction (refer to (iv)). Since it can exist at most one eventual missing edge, in finite time the robots of $\mathcal{S}_r$ succeed to reach node u, and stay on node u (refer to (iv)). Necessarily, $|\mathcal{S}_r| < \mathcal{R} - 2$, otherwise Rule $\mathbf{Term_2}$ is executed. At the time at which all the robots of $\mathcal{S}_r$ are on node u, by $(*)$, we know that at least one *awareSearcher*, on a node v, points to the right direction. By (vi), none of the *righter* of $\mathcal{S}_r$ can be located on node v. Therefore, this *awareSearcher* is not on node u. By the movements described in (iii) and (vii), we know that in finite time an *awareSearcher* succeeds to reach node u. Then all the *righter* of $\mathcal{S}_r$ become *awareSearcher*, hence by $(**)$, the lemma is proved.

Case 2.2: $\mathbf{x \neq u}$.

None of the robots of $\mathcal{S}_r$ can be located on the same node as an *awareSearcher*, otherwise, by (vi), they become *awareSearcher*. Therefore, none of the robots of $\mathcal{S}_r$ can be located on $Seg(u, x) \cup \{x\}$, otherwise, in finite time, by (iv) they are located on node x. However, once all the robots of $\mathcal{S}_r$ are on node x, by $(*)$, and the movements described in (iii) and (vii) an *awareSearcher* succeeds to be located on node u in finite time, which leads to a contradiction. Therefore all the robots of $\mathcal{S}_r$ are on nodes in $Seg(x, u)$. Since it can exist only one eventual missing edge, and since this edge is the adjacent right edge of x, for each robot r" of $\mathcal{S}_r$, by (iv), it exists a time at which r" is on node u while there is a present adjacent right edge to u. Therefore, by (v), in finite time all the robots of $\mathcal{S}_r$ are *awareSearcher* robots. Hence, by $(**)$, the lemma is proved.

We just proved that it exists a time t_{tower} at which a $towerMinConfiguration$ is present in the execution. We now prove that, at time t_{tower}, among the robots not involved in the $towerMinConfiguration$, there is at most one *righter*. By Lemma 6.6, there is only one $towerMinConfiguration$ in the whole execution. Necessarily, as explained above when there is no *potentialMin* in the execution, in order to have a $towerMinConfiguration$, a *righter* must become an *awareSearcher* while executing Rule $\mathbf{K_4}$. The property $(*)$ is then true. By definition of a $towerMinConfiguration$, only two robots are not involved in the $towerMinConfiguration$.

Assume, by contradiction, that there are two *righter* not involved in the $towerMinConfiguration$ at time t_{tower}. By $(*)$, this implies that there is an *awareSearcher* at time t_{tower}. However, by definition, a $towerMinConfiguration$ is composed of one $minWaitingWalker$ and $\mathcal{R}-3$ $waitingWalker$, therefore, since there are $\mathcal{R}$ robots in the system and among them, at time t_{tower}, two are *righter* and one is an *awareSearcher*, there is a contradiction with the fact that there is a $towerMinConfiguration$ at time t_{tower}. □

By Lemmas 6.15 and 6.16, we can deduce the following corollary which proves the correctness of Phase K.

Corollary 6.5. *There exists a time t in the execution at which a $towerMinConfiguration$ is present and among the robots not involved in the $towerMinConfiguration$ there is at most one righter robot at time t.*

Correctness Proof of Phases W and T

The combination of Phases W and T of $\mathbb{GDG}$ permit to solve $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. Since $\mathbb{G}_{EW}$ is divided into a safety and a liveness property, to prove the correctness of Phases W and T, we have to prove each of these two properties. We recall that, to satisfy the safety property of the gathering problem, all the robots that terminate their execution have to do so on the same node, and to satisfy the liveness property of $\mathbb{G}_{EW}$, at least $\mathcal{R}-1$ robots must terminate their execution in finite time. We first prove, in Lemma 6.19, that $\mathbb{GDG}$ satisfies the safety of the gathering problem in $\mathcal{COT}$ rings. Then, we prove, in Lemma 6.21, that $\mathbb{GDG}$ satisfies the liveness of $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. To prove these two properties, we need to prove some other lemmas.

By Corollary 6.5, we know that, in finite time, a $towerMinConfiguration$ is formed. By Lemma 6.6, there is at most one $towerMinConfiguration$ in the execution. Therefore, there is one and only one $towerMinConfiguration$ in the execution. Call T such a $towerMinConfiguration$. Let t_{tower} be the time at which T is formed. By definition, a $towerMinConfiguration$ is composed of $\mathcal{R}-2$ robots. If among the robots that are not involved in T one is a *righter* or a $potentialMin$, then call it r_1, and call r_2 the other robot not involved in T. If among the two robots that are not involved in T none of them are *righter* or $potentialMin$, then call one of them r_1, and the other one r_2.

In the previous section, we prove that, at time t_{tower}, at most one of the robots among the two robots not involved in T is a *righter*. In the following lemma, we go farther and give the set of possible values for the variable *kind* at time t_{tower} of each of these robots.

Lemma 6.17. *At time t_{tower}, $kind_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $kind_{r_2} \in \{dumbSearcher, awareSearcher\}$.*

Proof. Until the Look phase of time t_{tower}, by the rules of $\mathbb{GDG}$ and knowing that all the robots are initially *righter*, there are only *righter*, $potentialMin$, *dumbSearcher*, *awareSearcher*, $waitingWalker$ and $minWaitingWalker$ robots in the execution.

By Corollary 6.4, only r_{min} can be *min*, therefore only r_{min} can be $minWaitingWalker$. By definition of a $towerMinConfiguration$, a $minWaitingWalker$ is involved in T. Since r_1 and r_2 are not involved in T, this implies that neither r_1 nor r_2 can be $minWaitingWalker$ at time t_{tower}.

By definition of a $towerMinConfiguration$, at time t_{tower}, the $\mathcal{R}-2$ robots involved in T are on a same node. This node is the node u. Therefore, at time t_{tower} neither r_1 nor r_2 can be located on node u, otherwise Rule **Term₂** is executed. By Lemma 6.7, this implies that neither r_1 nor r_2 can be a $waitingWalker$ at time t_{tower}.

By Corollary 6.5, at time t_{tower}, only one robot among r_1 and r_2 can be a *righter* robot. Assume without lost of generality that r_1 is a *righter* at time t_{tower}. In this case by Corollary 6.5, r_2 cannot be a *righter* at time t_{tower}. Moreover, in this case, by Lemma 6.10, r_2 cannot be a $potentialMin$ at time t_{tower}.

Now assume without lost of generality that r_1 is a $potentialMin$ robot at time t_{tower}. By Lemmas 6.8 and 6.10, r_2 can neither be a $potentialMin$ nor a $righter$ at time t_{tower}.

This proves the lemma. □

In the following lemma, we prove a property on Rules **Term$_1$** and **Term$_2$** that helps us to prove that $\mathbb{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings.

Lemma 6.18. *If a robot r, on a node x, at a time t, executes Rule **Term$_1$** (resp. **Term$_2$**), then there are $\mathcal{R}$ (resp. $\mathcal{R}-1$) robots on node x at time t and they all execute Rule **Term$_1$** (resp. **Term$_2$**) at time t (if they are not already terminated).*

Proof. If a robot r, on a node x, executes Rule **Term$_1$** (resp. **Term$_2$**) at a time t, by the predicate $GE()$ (resp. $GEW()$), there are $\mathcal{R}$ (resp. $\mathcal{R}-1$) robots on x at time t. Moreover, if the predicate $GE()$ (resp. $GEW()$) is true for r at time t, since the robots are fully-synchronous, it is necessarily true for all the robots (not already terminated) located on node x at time t. This implies that all the robots (not already terminated) located on x at time t, execute Rule **Term$_1$** (resp. **Term$_2$**) at time t. □

Now we prove one of the main lemmas of this section: we prove that $\mathbb{GDG}$ solves the safety property of the gathering problem in $\mathcal{COT}$ rings.

Lemma 6.19. $\mathbb{GDG}$ *solves the safety of the gathering problem in $\mathcal{COT}$ rings.*

Proof. We want to prove that, while executing $\mathbb{GDG}$, all robots that terminate their execution terminate it on the same node. While executing $\mathbb{GDG}$, the only way for a robot to terminate its execution is to execute either Rule **Term$_1$** or Rule **Term$_2$**.

By Lemma 6.18, if a robot r, on a node x, at a time t, executes Rule **Term$_1$**, then there are $\mathcal{R}$ robots on node x at time t and they all execute Rule **Term$_1$** at time t (if they are not already terminated). Therefore, in the case where r executes Rule **Term$_1$** at time t, all the robots of the system are terminated on x at time t, hence the lemma is proved in this case.

By Lemma 6.18, if a robot r, on a node x, at a time t, executes Rule **Term$_2$**, then there are $\mathcal{R}-1$ robots on node x at time t and they all execute Rule **Term$_2$** at time t (if they are not already terminated). Therefore, in the case where r executes Rule **Term$_2$** at time t, $\mathcal{R}-1$ robots of the system are terminated on x at time t. Call r' the robot that is not on the node x at time t. Let y ($y \neq x$) be the node where r is located at time t. To prove the lemma, it stays to prove that r' is not terminated at time t, and that after time t, r' either terminates its execution on node x or never terminates its execution.

Assume, by contradiction, that at time t, r' is terminated. This implies that there exists a time $t' \leq t$ at which r' executes either Rule **Term$_1$** or Rule **Term$_2$**. By Lemma 6.18, this implies that at least $\mathcal{R}-2$ other robots are terminated on node y at time t'. Therefore, there is a contradiction with the fact that r executes Rule **Term$_2$** at time t on node x. Indeed, to execute Rule **Term$_2$** at time t on node x, $\mathcal{R}-1$ robots must be located on node x at time t, since $\mathcal{R} \geq 4$, it is not possible to have $\mathcal{R}-1$ robots on node x at time t.

Moreover, after time t, by Lemma 6.18, r' can terminate its execution only on node x (since it is the only node where $\mathcal{R}-1$ robots are located). Therefore, the lemma is proved. □

The following lemma is an extension of Lemma 6.13. While Lemma 6.13 is true when the robots are either executing Phase `M` or Phase `K`, the following lemma is true whatever the phase of the algorithm the robots are executing.

Lemma 6.20. *If, at time t, an isolated robot r is such that $kind_r \in \{dumbSearcher, awareSearcher\}$, then, during the Move phase of time $t-1$, it does not point to the $\perp$ direction.*

Proof. By the rules of $\mathbb{GDG}$, $minWaitingWalker$, $waitingWalker$, $minTailWalker$, $tailWalker$, $headWalker$ and $leftWalker$ cannot become $dumbSearcher$ or $awareSearcher$.

Consider an isolated robot r such that, at a time t, $kind_r \in \{dumbSearcher, awareSearcher\}$. Consider then the two following cases.

Case 1: At time t − 1, r is neither a *dumbSearcher* nor an *awareSearcher*.

Whatever the value of the variable $kind$ of r at time $t-1$ ($righter$ or $potentialMin$), to have its variable $kind$ at time t equals either to $dumbSearcher$ or to $awareSearcher$, r executes at time $t-1$ either Rule $\mathbf{K_4}$, $\mathbf{M_2}$, $\mathbf{M_3}$, $\mathbf{M_5}$, $\mathbf{M_6}$ or $\mathbf{M_7}$.

When a robot executes Rule $\mathbf{M_2}$, it calls the function BecomeAwareSearcher(_). When a robot executes the function BecomeAwareSearcher(_), it sets its direction to the $right$ direction, therefore the lemma is true in this case.

A robot executes Rule $\mathbf{M_3}$ only if it is located with a $headWalker$ on a node x. Necessarily there is no present adjacent right edge to x at time $t-1$, otherwise the robot would have executed Rule $\mathbf{M_2}$. By the rules of $\mathbb{GDG}$, a $headWalker$ only points to the $\perp$ direction or the right direction. Indeed, a $headWalker$ can only execute Rules $\mathbf{T_2}$, $\mathbf{T_3}$ and $\mathbf{W_1}$. While executing Rule $\mathbf{T_2}$, a $headWalker$ becomes a $leftWalker$ and points to the $\perp$ direction. While executing Rule $\mathbf{T_3}$, a $headWalker$ points to the $\perp$ direction. Finally, while executing Rule $\mathbf{W_1}$, a $headWalker$ points either to the right direction or to the $\perp$ direction. Therefore, even if, after the execution of Rule $\mathbf{M_3}$, r points to the $\perp$ direction, it is not isolated at time t, hence the lemma is not false in this case.

Then, we can use the arguments of the proof of Lemma 6.13 (in the case where the robot r is a $dumbSearcher$ or an $awareSearcher$ at time t) to prove that the current lemma is true for the remaining cases. Indeed, even if in Lemma 6.13 the context is such that there is no $towerMinConfiguration$ in the execution, the arguments used in its proof are still true in the context of the current lemma.

Case 2: At time t − 1, r is a *dumbSearcher* or an *awareSearcher*.

Whatever the value of the variable $kind$ of r at time $t-1$ ($dumbSearcher$ or $awareSearcher$), to have its variable $kind$ at time t equals either to $dumbSearcher$ or to $awareSearcher$, r executes at time $t-1$ either Rule $\mathbf{M_2}$, $\mathbf{M_3}$, $\mathbf{M_9}$, $\mathbf{M_{10}}$ or $\mathbf{M_{11}}$.

We can use the arguments of Case 1 to prove that while executing Rule $\mathbf{M_2}$ or $\mathbf{M_3}$, the lemma is proved.

Then, similarly as for the Case 1, we can use the arguments of the proof of Lemma 6.13 (in the case where the robot r is a $dumbSearcher$ or an $awareSearcher$ at time t) to prove that the current lemma is true in the remaining cases of Case 2.

□

Finally, we prove that $\mathbb{GDG}$ satisfies the liveness of $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings.

Lemma 6.21. *$\mathbb{GDG}$ satisfies the liveness of $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings.*

Proof. By contradiction, assume that $\mathbb{GDG}$ does not solve the liveness of $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. Since the execution of Rules $\mathbf{Term_1}$ and $\mathbf{Term_2}$ permits a robot to terminate its execution, by Lemma 6.18, this implies that there exists a $\mathcal{COT}$ ring such that, during the execution of $\mathbb{GDG}$, neither Rule $\mathbf{Term_1}$ nor Rule $\mathbf{Term_2}$ is executed. Consider the execution of $\mathbb{GDG}$ in that ring.

By Corollary 6.5, there exists a time t at which a $towerMinConfiguration$ is formed. Note that $\mathcal{R} - 2 \geq 2$ robots are involved in a $towerMinConfiguration$. Once a $towerMinConfiguration$ is formed the $\mathcal{R} - 3$ $waitingWalker$ and the $minWaitingWalker$ involved in this $towerMinConfiguration$ execute Rule $\mathbf{K_1}$. While executing this rule, the robot r with the

maximum identifier among the $\mathcal{R}-2$ robots involved in this $towerMinConfiguration$ becomes $headWalker$, the $minWaitingWalker$ becomes $minTailWalker$ and the other robots involved in this $towerMinConfiguration$ become $tailWalker$. Note that, by Corollary 6.4, only r_{min} can be min, and therefore, since r_{min} is the robot with the minimum identifier among all the robots of the system and since at least 2 robots are involved in the $towerMinConfiguration$, r_{min} cannot become $headWalker$. By Lemma 6.6 and by the rules of $\mathbb{GDG}$, only r can be $headWalker$ and only r_{min} can be $minTailWalker$ during the execution.

There is no rule in $\mathbb{GDG}$ permitting a $tailWalker$ or a $minTailWalker$ robot to become another kind of robot. A $tailWalker$ and a $minTailWalker$ can only execute Rules $\mathbf{T_3}$ and $\mathbf{W_1}$. By the rules of $\mathbb{GDG}$, the $minTailWalker$ and the $tailWalker$ execute the same movements at the same time starting from the same node, therefore, they are on a same node at each instant time. Hence, call $tail$ the set of all of these robots.

A $headWalker$ can become a $leftWalker$. However, since we assume that the liveness of $\mathbb{G}_{EW}$ cannot be solved, then it is not possible for r to become a $leftWalker$. Indeed, a $headWalker$ can only execute Rules $\mathbf{T_2}$, $\mathbf{T_3}$ and $\mathbf{W_1}$. Note that, by the rules of $\mathbb{GDG}$, after the execution of Rule $\mathbf{K_1}$, the $headWalker$ and the $tail$ both execute Rule $\mathbf{W_1}$. Therefore, since the $headWalker$ and the $tail$ start the execution of Rule $\mathbf{W_1}$ on the same node at the same time, by the rules of $\mathbb{GDG}$, while the $headWalker$ is executing Rule $\mathbf{T_3}$ or Rule $\mathbf{W_1}$, if the $tail$ is not on the same node as the $headWalker$, it is either executing Rule $\mathbf{W_1}$ or it is terminated. Moreover, by the same arguments, in the remaining of the execution, the $headWalker$ and the $tail$ are either on a same node or the $tail$ is on the left adjacent node (on the footprint of the dynamic ring) of the node where the $headWalker$ is located. Hence, if at a time t', the $headWalker$ executes Rule $\mathbf{T_2}$, and therefore becomes a $leftWalker$, then this implies that during time $t'-1$ it is executing either Rule $\mathbf{T_3}$ or Rule $\mathbf{W_1}$ while there is an adjacent left edge to its position and at time t' the $tail$ is not on its node. Therefore, necessarily the $tail$ is terminated, otherwise as explained the $tail$ would have joined the $headWalker$ on its node (Rule $\mathbf{W_1}$). Since only Rules $\mathbf{Term_1}$ and $\mathbf{Term_2}$ permit a robot to terminate its execution, by Lemma 6.18, this implies that the $tail$ has executed Rule $\mathbf{Term_2}$, which leads to a contradiction with the fact that $\mathbb{GDG}$ does not solve the liveness of $\mathbb{G}_{EW}$.

Therefore, during the whole execution (after the execution of Rule $\mathbf{K_1}$), the $headWalker$, $tailWalker$ and $minTailWalker$ stay respectively $headWalker$, $tailWalker$ and $minTailWalker$ and can only execute Rule $\mathbf{W_1}$ until their variables $walkSteps$ reach n, and then they can only execute Rule $\mathbf{T_3}$.

Call r_1 and r_2 the two robots that are not involved in the $towerMinConfiguration$ at time t. Since, by contradiction, neither Rule $\mathbf{Term_1}$ nor Rule $\mathbf{Term_2}$ are true, neither r_1 nor r_2 can meet the $headWalker$ or the $tail$ while they (the $headWalker$ and the $tail$) are on a same node. Therefore, we assume that this event never happens.

By Lemma 6.17, at time t, $kind_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $kind_{r_2} \in \{dumbSearcher, awareSearcher\}$.

Let us first consider all the possible interactions between only r_1 and r_2 while $kind_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $kind_{r_2} \in \{dumbSearcher, awareSearcher\}$.

An isolated $potentialMin$ or a $potentialMin$ that is located only with a $dumbSearcher$ stays a $potentialMin$ and points to the $right$ direction (Rule $\mathbf{M_8}$).

If a $potentialMin$ is located only with an $awareSearcher$, it becomes an $awareSearcher$ and it executes the function SEARCH() (Rule $\mathbf{M_5}$).

An isolated $righter$ stays a $righter$ and points to the $right$ direction (Rule $\mathbf{M_8}$).

If a $righter$ is located only with a $dumbSearcher$ (resp. an $awareSearcher$), it becomes an $awareSearcher$ and executes the function SEARCH() (Rule $\mathbf{M_7}$).

If a $dumbSearcher$ is located only with a $righter$, it becomes an $awareSearcher$ and executes the function SEARCH() (Rule$\mathbf{M_9}$).

If a *dumbSearcher* is located only with a *potentialMin* it stays a *dumbSearcher* and executes the function SEARCH() (Rule $\mathbf{M_{11}}$). In this case, while executing the function SEARCH(), a *dumbSearcher* points to the $left$ direction, since it possesses a greater identifier than the one of the *potentialMin*. Indeed, only Rule $\mathbf{M_6}$ permits a robot to become *potentialMin* or *dumbSearcher*. This rule is executed when $\mathcal{R} - 1$ *righter* are located on a same node. While executing Rule $\mathbf{M_6}$, among the $\mathcal{R} - 1$ *righter*, the one with the minimum identifier becomes *potentialMin* while the others become *dumbSearcher*. By Observation 6.2, Rule $\mathbf{M_6}$ can be executed only once. Therefore, a *dumbSearcher* necessarily possesses an identifier greater than the one of the *potentialMin*.

An isolated *dumbSearcher* or a *dumbSearcher* located only with another *dumbSearcher* stays a *dumbSearcher* and executes the function SEARCH() (Rule $\mathbf{M_{11}}$).

If a *dumbSearcher* is located only with an *awareSearcher*, it becomes an *awareSearcher* and it executes the function SEARCH() (Rule $\mathbf{M_{10}}$).

An isolated *awareSearcher* or an *awareSearcher* located only with a *righter*, a *potentialMin*, a *dumbSearcher* or an *awareSearcher* stays an *awareSearcher* and executes the function SEARCH() (Rule $\mathbf{M_{11}}$).

When r_1 and r_2 are on a same node without any other robot, executing the function SEARCH(), since all the robots possess distinct identifiers, one points to the *right* direction, while the other one points to the $left$ direction.

While executing the function SEARCH() at time i, a robot that is an isolated *dumbSearcher* or an isolated *awareSearcher* points during the Move phase of time i to the same direction it points to during the Move phase of time $i - 1$. By Lemma 6.20, this direction cannot be equal to $\perp$.

By the previous movements described, note that, as long as r_1 and r_2 are not located with the *headWalker* or the *tail*, they are always such that $kind_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $kind_{r_2} \in \{dumbSearcher, awareSearcher\}$.

Now, consider the possible interactions between the *headWalker* and r_1 and/or r_2 when $kind_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $kind_{r_2} \in \{dumbSearcher, awareSearcher\}$.

If r_1 and/or r_2, as a *righter*, *potentialMin*, *dumbSearcher* or *awareSearcher* is on the same node as the *headWalker* such that there is no adjacent *right* edge to their location, then it executes Rule $\mathbf{M_3}$, hence it becomes an *awareSearcher* and stops to move.

($*$) If r_1 and/or r_2, as a *righter*, *potentialMin*, *dumbSearcher* or *awareSearcher* is on the same node as the *headWalker* such that there is an adjacent *right* edge to their location, then it executes Rule $\mathbf{M_2}$, hence it becomes an *awareSearcher* pointing to the right direction and therefore crosses the adjacent right edge to its node.

This implies that, as long as r_1 and r_2 are not located with the *tail* they are always such that $kind_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $kind_{r_2} \in \{dumbSearcher, awareSearcher\}$.

Finally, consider the possible interaction between the *tail* and r_1 and/or r_2. If r_1 and/or r_2, as a *righter*, *potentialMin*, *dumbSearcher* or *awareSearcher* is on the same node as the *minTailWalker*, then it executes Rule $\mathbf{M_4}$ and becomes a *tailWalker*. From this time, by the function BECOMETAILWALKER(_) and the rules of $\mathbb{GDG}$, the robot belongs to the *tail*.

We consider two cases: the case where there exists an eventual missing edge, and the case where there is no eventual missing edge during the execution.

First, we assume that there exists an eventual missing edge. Call t" the time after the execution of Rule $\mathbf{K_1}$ and after the time when the eventual missing edge is missing forever. Consider the execution from t". Since Rule $\mathbf{K_1}$ is executed before time t", then there are *headWalker*, *tailWalker* and *minTailWalker* in the execution after time t" included.

Recall that, while executing Rules $\mathbf{W_1}$ and $\mathbf{T_3}$, the *headWalker* and the *tail* are either on a same node or on two adjacent nodes (the *tail* is on the adjacent left node on the footprint of the dynamic ring of the node where the *headWalker* is located).

Case 1: There is an eventual missing edge e between the node where the headWalker is located and the node where the tail is located.

As explained previously, since the *headWalker* and the *tail* are not on the same node, this necessarily implies that the *headWalker* either executes Rule $\mathbf{W_1}$ or Rule $\mathbf{T_3}$ at time t", and the *tail* executes Rule $\mathbf{W_1}$ at time t". Therefore, after time t", the *headWalker* does not move either because it waits for the *tail* to join it on its node (Rule $\mathbf{W_1}$), or because it executes the function StopMoving() (Rule $\mathbf{T_3}$). Similarly, after time t", the *tail* does not move, since it tries to join the *headWalker* pointing to the right direction (Rule $\mathbf{W_1}$), but the edge is missing forever.

Since there is at most one eventual missing edge in a $\mathcal{COT}$ ring, all the edges, except e, are infinitely often present in the execution after time t". Considering the movements of the robots described previously, whatever the direction pointed to by r_1 and r_2 at time t" both of them succeed eventually to reach the node where the *tail* is located, making the liveness of $\mathbb{G}_{EW}$ solved.

Case 2: The eventual missing edge is not between the node where the headWalker is located and the node where the tail is located.

This implies that there exists a time from which the *headWalker* and the *tail* are located on a same node and do not move, either because they are executing Rule $\mathbf{T_3}$, or because they are executing Rule $\mathbf{W_1}$ but the adjacent *right* edge the *headWalker* tries to cross is the eventual missing edge. In the second case, by the movements of the robots described previously, we succeed to prove that, eventually at most one of the robots among r_1 and r_2 can be stuck on the extremity of the eventual missing edge where the *headWalker* and the *tail* are not located, and that at least one of them succeeds to reach the node where the *headWalker* and the *tail* are located, making the liveness of $\mathbb{G}_{EW}$ solved.

Consider now the first case. Call t_n the first time at which the *headWalker* and the *tail* are on a same node and both execute Rule $\mathbf{T_3}$. If r_1 and r_2 point to the same direction at time t_n, then by the movements of the robots described previously, whatever the place of the eventual missing edge, we succeed to prove that, eventually at most one of them can be stuck on one of the extremities of the eventual missing edge, and that at least one of them succeeds to reach the node where the *headWalker* and the *tail* are located, making the liveness of $\mathbb{G}_{EW}$ solved. Similarly, if the *headWalker* and the *tail* are located, at time t_n, on one of the extremities of the eventual missing edge, then, by the movements of the robots described previously, we succeed to prove that, eventually at most one of the robots among r_1 and r_2 can be stuck on the extremity of the eventual missing edge where the *headWalker* and the *tail* are not located, and that at least one of them succeeds to reach the node where the *headWalker* and the *tail* are located, making the liveness of $\mathbb{G}_{EW}$ solved.

Now consider the first case, when r_1 and r_2 point to opposed directions at time t_n and such that, at time t_n, the *headWalker* and the *tail* are not located on one of the extremities of the eventual missing edge. It is not possible for both r_1 and r_2 to be eventually stuck on two different extremities of the eventual missing edge. Indeed, if r_1 and r_2 point to two opposed directions at time t_n, this is because, between times t_i and t_n (with t_i the time at which the *headWalker* and the *tail* both execute Rule $\mathbf{W_1}$ for the first time), they are located on a same node (without any other robot on their node). We prove this by contradiction. Assume, by contradiction, that r_1 and r_2 are never located on a same node (without any other robot on their node) between times t_i and t_n. Consider the execution from time t_i until time t_n. Whatever the direction pointed to by r_1 (resp. r_2), it cannot be located with the *tail*, otherwise, since there is no eventual missing edge between the *headWalker* and the *tail* and by the movements of the robots described previously, Rule $\mathbf{Term_2}$ is eventually executed. Therefore, r_1 (resp. r_2) can only be located with the *headWalker*. When r_1

(resp. r_2) is located with the $headWalker$, it necessarily exists an adjacent right edge to their position before the adjacent left edge to their position appears, otherwise, the *tail* joins them and Rule **Term$_2$** is executed. By $(*)$, after r_1 (resp. r_2) is on the same node as the $headWalker$ while there is an adjacent *right* edge to their location, it becomes an $awareSearcher$ pointing to the right direction. At time t_n, the $headWalker$ and the *tail* execute Rule **T$_3$**, therefore they succeed to execute Rule **W$_1$** until their variables $walkSteps$ is equal to n. This implies that, if r_1 (resp. r_2) points to the left direction at time t_i, necessarily, since it cannot be located with r_2 (resp. r_1), by the movements of the robots described previously, it exists a time $t_{meet} \geq t_i$ at which the $headWalker$ and the *tail* execute Rule **W$_1$** and either the $headWalker$ or the *tail* is located with it. As explained previously, r_1 (resp. r_2) cannot be located with the *tail*, this implies that, at time t_{meet}, r_1 (resp. r_2) is located with the $headWalker$. Therefore, whatever the direction pointed to by r_1 (resp. r_2) at time t_i, if r_1 and r_2 are never located on a same node (without any other robot on their node) between times t_i and t_n, it necessarily points to the right direction at time t_n. Indeed, r_1 (resp. r_2) points to the right direction at time t_n either because it meets the $headWalker$ that makes it point to the right direction or because at time t_i it points to the right direction and it is never located with the $headWalker$ and, by the movements of the robots described previously, it has not changed its direction between times t_i and time t_n. Hence, there is a contradiction with the fact that r_1 and r_2 point to opposite directions at time t_n. Therefore, r_1 and r_2 point to two opposite directions at time t_n because they are located on a same node (without any other robot on their node) between times t_i and t_n.

Consider the last time t_l between times t_i and t_n at which r_1 and r_2 are located on a same node (without any other robot on their node). At time t_l, since the two robots are located on a same node, by the movements of the robots described, during the Move phase of time t_l one points to the right direction while the other one points to the left direction. By assumption, between times $t_l + 1$ and t_n, r_1 and r_2 are not located on a same node. Moreover, as explained previously, between times $t_l + 1$ and t_n, neither r_1 nor r_2 can be located with the *tail*, otherwise Rule **Term$_2$** is eventually executed. Besides, between times $t_l + 1$ and t_n the robot that points to the left direction during the Move phase of time t_l cannot be located with the $headWalker$, otherwise, as noted previously, it points to the right direction at time t_n. Similarly, it is not possible for the robot that points to the *right* direction during the Move phase of time t_l to be located with the $headWalker$ between times $t_l + 1$ and t_n, otherwise, by the movements of the robots described previously, this necessarily implies that either it is also located on the same node as the *tail* and therefore the liveness of $\mathbb{G}_{EW}$ is solved or r_1 and r_2 are on a same node and therefore the robot that points to the left direction during the Move phase of time t_l is located with the $headWalker$. Therefore, from time $t_l + 1$ to time t_n, r_1 and r_2 are isolated, hence, by the movements of the robots, they point to the same respective directions from the Move phase of time t_l to time t_n.

Assume, without lost of generality, that this is r_1 that points to the right direction from the Move phase of time t_l to time t_n. Call v_1 (resp. v_2) the node on which r_1 (resp. r_2) is located at time t_n. The explanations of the previous paragraph imply that $v_1 \neq v_2$, and that, at time t_n, the node where the $headWalker$ and the *tail* are located is in $Seg(v_1, v_2)$. Therefore, since r_1 (resp. r_2) points to the right (resp. the left) direction at time t_n, by the movements of the robots and since it exists only one eventual missing edge, this is not possible for these two robots to be eventually stuck on each of the extremities of the eventual missing edge. Hence, at least one succeeds to reach the node where the $headWalker$ and the *tail* are located, making the liveness of $\mathbb{G}_{EW}$ solved.

Now we consider the case where there is no eventual missing edge. In this case, as indicated, it is not possible for both r_1 and r_2 to join the *tail* otherwise Rule **Term$_2$** is executed, which leads

to a contradiction with the assumption that $\mathbb{GDG}$ does not satisfy the liveness of $\mathbb{G}_{EW}$. Similarly, if one of the robots among r_1 or r_2 joins the *tail*, then, since there is no eventual missing edge, eventually the liveness of $\mathbb{G}_{EW}$ is satisfied (Rule $\mathbf{Term_2}$ is executed): either the robot joins the *tail* while the *tail* is located on the same node as the $headWalker$, or the robot joins the *tail*, executes Rule $\mathbf{M_4}$, and then the *tail* (that includes the robot that have executed Rule $\mathbf{M_4}$) join the $headWalker$. Hence, none of the robots among r_1 and r_2 can join the *tail*. By the same arguments, it is not possible for the *tail* to join the $headWalker$ while there is on its node at least one of the robots among r_1 and r_2. This implies, since there is no eventual missing edge, that the $headWalker$ and the *tail* succeed to execute Rule $\mathbf{W_1}$ until their variables $walkSteps$ reach n, and then they only execute Rule $\mathbf{T_3}$ forever. Call n_s the node on which the $headWalker$ and the *tail* execute Rule $\mathbf{T_3}$, and hence the node on which they stop to move forever. At the time t_s where both the $headWalker$ and the *tail* execute Rule $\mathbf{T_3}$, r_1 and r_2 are such that $kind_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $kind_{r_2} \in \{dumbSearcher, awareSearcher\}$. By the movements of the robots described previously, since there is no eventual missing edge, at least one of the robots among r_1 and r_2 succeed to reach n_s in finite time after time t_s. Hence, in the case where there is no eventual missing edge, Rule $\mathbf{Term_1}$ and/or $\mathbf{Term_2}$ is executed. Therefore $\mathbb{G}_{EW}$ is also solved by $\mathbb{GDG}$ in this case. □

By Lemmas 6.19 and 6.21, we can deduce the following theorem which proves the correctness of Phases `W` and `T`.

Theorem 6.2. *$\mathbb{GDG}$ satisfies $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings under $\mathcal{M}_{gathering}$.*

6.3.2 What about $\mathbb{GDG}$ executed in $\mathcal{AC}$, $\mathcal{RE}$, $\mathcal{BRE}$ and $\mathcal{ST}$ rings?

In the previous section, we prove that $\mathbb{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. In this section, we consider $\mathcal{AC}$, $\mathcal{RE}$, $\mathcal{BRE}$ and $\mathcal{ST}$ rings. For each of these classes of dynamic rings, we give the version of gathering $\mathbb{GDG}$ solves in it (refer to Table 6.1). When $\mathbb{GDG}$ solves a bounded version of the gathering problem, we give its time complexities in rounds.

First, we consider the case of $\mathcal{RE}$ rings. In the following theorem, we prove that $\mathbb{GDG}$ solves $\mathbb{G}_E$ in $\mathcal{RE}$ rings.

Theorem 6.3. *$\mathbb{GDG}$ satisfies $\mathbb{G}_E$ in $\mathcal{RE}$ rings under $\mathcal{M}_{gathering}$.*

Proof. By Theorem 6.2, $\mathbb{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings, therefore it solves the safety and the liveness of $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. Since $\mathcal{RE} \subset \mathcal{COT}$, $\mathbb{GDG}$ also solves the safety and the liveness of $\mathbb{G}_{EW}$ in $\mathcal{RE}$ rings. This implies that all robots that terminate their execution terminate it on the same node and it exists a time at which at least $\mathcal{R} - 1$ robots terminate their execution. Call t the first time at which at least $\mathcal{R} - 1$ robots terminate their execution.

By contradiction, assume that $\mathbb{GDG}$ does not solve $\mathbb{G}_E$ in $\mathcal{RE}$ rings, this implies that it exists a robot r that never terminates its execution.

Call $towerTermination$ the $\mathcal{R} - 1$ robots that, at time t, are located on a same node and are terminated. While executing $\mathbb{GDG}$, the only way for a robot to terminate its execution is to execute either Rule $\mathbf{Term_1}$ or Rule $\mathbf{Term_2}$. By Lemma 6.18, for a $towerTermination$ to be formed at time t, Rule $\mathbf{Term_2}$ has to be executed at this time.

($*$) By the predicate of Rule $\mathbf{Term_2}$, r_{min} belongs to the $towerTermination$. By Lemma 6.18, all the robots that are located on the same node as r_{min} at time t belong to the $towerTermination$.

Call w the node where the $towerTermination$ is located at time t.

Note that r cannot be located on node w after time t included, otherwise it executes Rule $\mathbf{Term_1}$ and the lemma is proved.

Since r_{min} belongs to the $towerTermination$, and since by Corollary 6.4, only r_{min} can be $minWaitingWalker$ or a $minTailWalker$, r is neither $minWaitingWalker$ nor $minTailWalker$.

At time t, r cannot be a $tailWalker$. Indeed, to become a $tailWalker$, a robot must either execute Rule $\mathbf{K_1}$ or Rule $\mathbf{M_4}$. To execute Rule $\mathbf{K_1}$ a robot must be a $waitingWalker$. By Lemma 6.7, all $waitingWalker$ are located on the same node as the $minWaitingWalker$. Moreover, when a $waitingWalker$ executes Rule $\mathbf{K_1}$, by the predicate $AllButTwoWaitingWalker()$, the $minWaitingWalker$ also executes this rule becoming a $minTailWalker$. Then by the rules of $\mathbb{GDG}$, the robot that becomes $tailWalker$ while executing Rule $\mathbf{K_1}$ and the $minTailWalker$ execute the same movements (refer to Rules $\mathbf{W_1}$ and $\mathbf{T_3}$), and therefore are always on a same node. Besides, to execute Rule $\mathbf{M_4}$ a robot must be located on the same node as the $minWaitingWalker$ (refer to the predicate $NotWalkerWithTailWalker(r')$). Then, thanks to the function BecomeTailWalker(_) and by the rules of $\mathbb{GDG}$, the robot that becomes $tailWalker$ while executing Rule $\mathbf{M_4}$ cannot be on a node different from the one where the $minTailWalker$ is located (refer to Rules $\mathbf{W_1}$ and $\mathbf{T_3}$). Therefore, by $(*)$, r cannot be a $tailWalker$ at time t, otherwise, at time t, it is on the same node as the $minTailWalker$ (thus, by Corollary 6.4, it is on the same node as r_{min}) and hence it terminates its execution.

At time t, r cannot be a $waitingWalker$ robot. Indeed by the rules of $\mathbb{GDG}$ and the previous remarks, it cannot exist $waitingWalker$ if there is no $minWaitingWalker$ in the execution, and by Lemma 6.7 all the $waitingWalker$ and $minWaitingWalker$ are located on a same node. Therefore, by $(*)$, r cannot be a $waitingWalker$ at time t, otherwise, at time t, it is on the same node as the $minWaitingWalker$ (thus, by Corollary 6.4, it is on the same node as r_{min}) and hence it terminates its execution.

Therefore, at time t, r can be either a $righter$, a $potentialMin$, a $dumbSearcher$, an $awareSearcher$, a $headWalker$ or a $leftWalker$ robot.

As long as r is not on node w, it is isolated.

An isolated $righter$ or an isolated $potentialMin$ only executes Rule $\mathbf{M_8}$. While executing this rule, a robot points to the $right$ direction and stays a $righter$ or a $potentialMin$. Since all the edges are infinitely often present, such a robot is infinitely often able to move in the $right$ direction until reaching the node w.

An isolated $dumbSearcher$ or an isolated $awareSearcher$ only executes Rule $\mathbf{M_{11}}$. While executing this rule, an isolated robot stays a $dumbSearcher$ or an $awareSearcher$, and points to the direction it points to during the previous Move phase. By Lemma 6.20, this direction cannot be equal to $\perp$. Therefore, an isolated $dumbSearcher$ or an isolated $awareSearcher$ always points to the same direction d (either $right$ or $left$). Since all the edges are infinitely often present, such a robot is infinitely often able to move in the direction d until reaching the node w.

Now assume that, at time t, r is a $leftWalker$. A $leftWalker$ only executes Rule $\mathbf{T_1}$. While executing this rule, a robot points to the $left$ direction and stays a $leftWalker$. Since all the edges are infinitely often present, such a robot is infinitely often able to move in the $left$ direction until reaching the node w.

Now assume that, at time t, r is a $headWalker$. A $headWalker$ can execute either Rule $\mathbf{T_2}$ or Rule $\mathbf{T_3}$ or Rule $\mathbf{W_1}$. While executing Rule $\mathbf{T_2}$, a $headWalker$ becomes a $leftWalker$, then, by the previous paragraph, r reaches the node w in finite time. Consider now the cases where, at time t, r executes either Rule $\mathbf{T_3}$ or Rule $\mathbf{W_1}$. In these cases, after time t, it necessarily exists a time at which r executes Rule $\mathbf{T_2}$. Assume, by contradiction, that this is not true. The only way for a robot to become a $headWalker$ is to execute Rule $\mathbf{K_1}$. Rule $\mathbf{K_1}$ is executed when $\mathcal{R}-2$ robots are located on a same node. While executing this rule, a robot sets its variable $walkerMate$ with the identifiers of the robots that are located on its node. Only Rule $\mathbf{K_1}$ permits a robot to update its variable $walkerMate$. Note that, since $\mathcal{R}-2 \geq 2$, the variable $walkerMate$ of r, after time t, contains at least one identifier i different from the identifier of r. The robot of identifier i necessarily belongs to the $towerTermination$, since only r does not terminate. (1) Hence, at time t, the robot of identifier i is terminated on node w, thus it does not move, and therefore, after time t, r is never on the same node as i. (2) All the edges are infinitely often present. While executing Rule $\mathbf{T_3}$ at time t, r points to the $\perp$ direction and does not update its

other variables. (3) Hence, by the rules of $\mathbb{GDG}$, since r cannot execute Rule $\mathbf{T_2}$, after time t, r can only execute Rule $\mathbf{T_3}$, and therefore only points to the $\perp$ direction. Hence, necessarily by (1), (2) and (3), this implies that, after time t, it exists a time at which the predicate $HeadWalker$-$WithoutWalkerMate()$ is true, thus at this time Rule $\mathbf{T_2}$ is executed. Similarly, if at time t, r executes Rule $\mathbf{W_1}$, since r can never be located on the same node as i, while executing Rule $\mathbf{W_1}$, it points to the $\perp$ direction and does not update its other variables. (4) Hence, by the rules of $\mathbb{GDG}$, since r cannot execute Rule $\mathbf{T_2}$, after time t, r can only execute Rule $\mathbf{W_1}$, and therefore always points to the $\perp$ direction. Thus, by (1), (2) and (4), necessarily, after time t, it exists a time at which the predicate $HeadWalkerWithoutWalkerMate()$ is true, hence at this time Rule $\mathbf{T_2}$ is executed. Therefore, even in the cases where, at time t, r executes either Rule $\mathbf{T_3}$ or Rule $\mathbf{W_1}$, it exists a time greater than t at which r becomes a $leftWalker$ and hence, by the previous paragraph, r succeeds to reach the node w in finite time.

Therefore whatever the kind of robot r is, it is always able to reach the node w. Once r reaches the node w it executes Rule $\mathbf{Term_1}$ making $\mathbb{G}_E$ solved. □

Now, we consider the case of $\mathcal{AC}$ rings. In the following theorem, we prove that $\mathbb{GDG}$ solves $\mathbb{G}_W$ in $\mathcal{AC}$ rings.

Theorem 6.4. *$\mathbb{GDG}$ satisfies $\mathbb{G}_W$ in $\mathcal{AC}$ rings under $\mathcal{M}_{gathering}$ in $O(id_{r_{min}} * n^2 + \mathcal{R} * n)$ rounds.*

Proof. By Theorem 6.2, $\mathbb{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings, since $\mathcal{AC} \subset \mathcal{COT}$, this implies that $\mathbb{GDG}$ also solves $\mathbb{G}_{EW}$ in $\mathcal{AC}$ rings. Therefore, to prove that $\mathbb{GDG}$ solves $\mathbb{G}_W$ in $\mathcal{AC}$ rings, it stays to prove that each phase of $\mathbb{GDG}$ is bounded.

Phase `M`: By Corollary 6.4, only r_{min} becomes min in finite time. By the rules of $\mathbb{GDG}$, when r_{min} becomes min, it is first $minWaitingWalker$ before being $minTailWalker$ (since only a $minWaitingWalker$ can become a $minTailWalker$ while executing Rule $\mathbf{K_1}$). Therefore, since only Rule $\mathbf{M_1}$ permits a robot to become $minWaitingWalker$, by the predicate $MinDiscovery()$ of this rule, r_{min} becomes min either because it moves during $4 * n * id_{r_{min}}$ steps in the right direction or because it meets a robot that permits it to deduce that it is min. In this last case, note that, either r_{min} is $potentialMin$, or r_{min} meets a $potential$-Min or a $dumbSearcher$ or a robot whose variable $idMin$ is different from -1. Therefore, in this last case, either r_{min} possesses a variable $idPotentialMin$ different from -1, or r_{min} meets a robot r such that $idPotentialMin_r$ is different from -1 (since a $potentialMin$ and a $dumbSearcher$ have their variable $idPotentialMin$ different from -1 (Rule $\mathbf{M_6}$) and since, while executing $\mathbb{GDG}$, each time the variable $idMin$ of a robot is set with a variable different from -1, this is also the case for its variable $idPotentialMin$).

Taking back the arguments used in the proof of Lemma 6.5, let us consider the following cases.

Case 1.1: Rule $\mathbf{M_6}$ is never executed.

By the rules of $\mathbb{GDG}$, this implies that, before the time when r_{min} is min, there are only $righter$ in the execution. First, this implies that r_{min} becomes min because it moves during $4 * n * id_{r_{min}}$ steps in the $right$ direction (since $righter$ robots have their variables $idPotentialMin$ equal to -1). Second, in this context, as long as r_{min} is not min, all the $righter$ always point to the $right$ direction (Rule $\mathbf{M_8}$). This implies that, as long as r_{min} is not min, each time a robot wants to move in the right direction it can be stuck during at most n rounds, otherwise, since in an $\mathcal{AC}$ ring at most one edge can be missing at each instant time, Rule $\mathbf{M_6}$ is executed. Therefore in case 1.1 r_{min} becomes min in at most $4 * id_{r_{min}} * n * n$ rounds.

Now let consider the case where Rule $\mathbf{M_6}$ is executed at a time t. In the following, we consider the execution from time t. After time t, while it is not yet min, if r_{min} is stuck

more than $4 * n$ consecutive rounds on a same node then it becomes min. We prove this considering the two following cases. In each of these cases, we assume that r_{min} is not yet min and that it is stuck more than n rounds on a same node.

Case 1.2: Rule $\mathbf{M_6}$ is executed but $\mathbf{r_{min}}$ is not among the $\mathcal{R}-1$ righter robots that execute it.

Taking back the arguments of the proof of Lemma 6.5, we know that Rule $\mathbf{M_6}$ can be executed only once, and that the robots that execute this rule can never be *righter* anymore. Moreover, since r_{min} does not execute Rule $\mathbf{M_6}$, since, by Corollary 6.4, r_{min} necessarily becomes min, since, by Lemma 6.9, only *righter* and $potentialMin$ can be min, and since only Rule $\mathbf{M_6}$ permits robots to become $potentialMin$, before becoming min, r_{min} is a *righter*. By the proof of Lemma 6.5, as long as r_{min} is not min it cannot exist *awareSearcher*. Hence, by the rules of $\mathbb{GDG}$, as long as r_{min} is not min, there are only one *righter*, one $potentialMin$ and $\mathcal{R}-2$ *dumbSearcher* in the execution. Therefore, by the rules of $\mathbb{GDG}$, the $potentialMin$ is $potentialMin$ at least until r_{min} becomes min. Hence, the $potentialMin$ executes Rule $\mathbf{M_8}$ and thus points to the right direction at least until r_{min} becomes min. We have assumed that, while it is not yet min, r_{min} is stuck more than $4*n$ consecutive rounds on a same node. Since r_{min} is a *righter* before being min, it is stuck because the adjacent right edge to its position is missing (Rule $\mathbf{M_8}$). Therefore, since in an $\mathcal{AC}$ ring of size n at least $n-1$ edges are present at each instant time, either the $potentialMin$ (or a *dumbSearcher*) meets r_{min} in at most n rounds. When r_{min} meets a $potentialMin$ (or a *dumbSearcher*), it becomes min by definition of the predicate $MinDiscovery()$ in Rule $\mathbf{M_1}$. Therefore, if it is stuck more than $4 * n$ rounds, r_{min} becomes min in at most n rounds.

Case 1.3: Rule $\mathbf{M_6}$ is executed and $\mathbf{r_{min}}$ is among the $\mathcal{R}-1$ righter robots that execute it.

In this case, by Rule $\mathbf{M_6}$, r_{min} becomes $potentialMin$. By Observations 6.2 and 6.1, by Corollary 6.4 and by Lemma 6.9 r_{min} is $potentialMin$ until it becomes min. Therefore, r_{min}, while it is not yet min, can be stuck only because the adjacent right edge to its position is missing (Rule $\mathbf{M_8}$).

First, consider that at the time when r_{min}, as a $potentialMin$, is stuck more than $4*n$ rounds, there does not exist *righter* in the execution. By Observation 6.2, there is no more *righter* in the execution. However, at the time when Rule $\mathbf{M_6}$ is executed, the robot r that is not among the robots that execute this rule is a *righter*. Therefore, necessarily r, as a *righter*, meets at least one *dumbSearcher* at a time t'. Indeed, it cannot meet the $potentialMin$, otherwise r_{min} is min (Rule $\mathbf{M_1}$), and thus it is not anymore $potentialMin$ at the time at which it is stuck. Moreover, r cannot be isolated forever after time t, otherwise it stays a *righter* (Rule $\mathbf{M_8}$). Hence, at time t', r becomes an *awareSearcher* (Rule $\mathbf{M_7}$). Consider an *awareSearcher* r_a of the execution. By Lemma 6.4, r_a cannot point to the $\perp$ direction. Moreover, by the rules of $\mathbb{GDG}$, as long as there is no min, an *awareSearcher* executes the function Search() (rule $\mathbf{M_{11}}$). Besides, by the proof of Lemma 6.5 if a robot is not isolated and executes the function Search(), then all the robots of its node are or become *awareSearcher* and execute the function Search(). While executing the function Search(), an isolated robot does not change its direction. When a robot executes the function Search() while there are multiple robots on its node, if it possesses the maximum identifier among the robots of its node, it points to the left direction, otherwise it points to the right direction. In an $\mathcal{AC}$ ring of size n, at least $n-1$ edges are present at each instant time. Therefore, if r_a points to the *right* direction, either it, as an *awareSearcher* or a robot that is or becomes an *awareSearcher* is located, in at most n rounds, on the node where r_{min}, as a $potentialMin$, is stuck. In the

case where r_a points to the $left$ direction then, by the same arguments, in at most $4 * n$ rounds an *awareSearcher* is located on the node where r_{min}, as a *potentialMin*, is stuck. Indeed, at most n rounds are needed for an *awareSearcher*to reach the extremity of the missing edge where r_{min} is not located. Then, at most $2 * n$ other rounds are needed for a *dumbSearcher* (execution of the function SEARCH(), rule $\mathbf{M_{11}}$) or an *awareSearcher* to reach also this node. These $2 * n$ rounds are especially needed for a *dumbSearcher* that may take n rounds (pointing to the left direction) to reach the node where r_{min} is stuck and then again n rounds (pointing to the right direction) to reach the other extremity of the missing edge. From this time there is in the execution an *awareSearcher* that points to the *right* direction. Finally, at most n supplementary rounds are needed for an *awareSearcher* to reach the node where r_{min}, as a *potentialMin*, is stuck. Note that $\mathcal{R} > 4$, and there are $\mathcal{R} - 1$ *dumbSearcher*/*awareSearcher* in the execution as long as r_{min} is not *min*. Therefore, the previous scenario can effectively happen. When r_{min} meets an *awareSearcher*, it becomes *min* by definition of the predicate *MinDiscovery*() of rule $\mathbf{M_1}$. Therefore, r_{min} becomes *min* in at most $4 * n$ rounds if it is stuck more than $4 * n$ rounds.

Second, consider that at the time when r_{min}, as a *potentialMin*, is stuck more than $4 * n$ rounds, there exists a *righter*. In this case, since an isolated *righter* points to the *right* direction (Rule $\mathbf{M_8}$), and by the arguments of the previous paragraph, either a *righter* or a robot that is an *awareSearcher* or that becomes an *awareSearcher*(Rules $\mathbf{M_7}$, $\mathbf{M_9}$ or $\mathbf{M_{10}}$) meets r_{min} in at most n rounds. When r_{min} meets a *righter* or an *awareSearcher*, it becomes *min* by definition of the predicate *MinDiscovery*() of Rule $\mathbf{M_1}$. Therefore, r_{min} becomes *min* in at most n rounds if it is stuck more than $4 * n$ rounds.

Now, we give the worst number of rounds needed for r_{min} to become *min*, in the case where there exists a time t at which Rule $\mathbf{M_6}$ is executed. By Case 1.1, before time t, r_{min}, while it is not yet *min*, can be stuck at most n rounds each time it moves from one step in the *right* direction. Similarly, by the two previous cases (Case 1.2 and 1.3), after time t, r_{min}, while it is not yet *min*, can be stuck at most $4 * n$ rounds each time it moves from one step in the *right* direction. Let nb be the number of steps in the right direction moved by r_{min} before time t. As proved previously, r_{min} is either a *righter* or a *potentialMin* before being *min*. By Lemma 6.2, this implies that before being *min*, r_{min} always points to the right direction. Therefore, by the predicate *MinDiscovery*() of Rule $\mathbf{M_1}$, in at most $nb * n + ((4 * id_{r_{min}} * n) - nb) * 4 * n$ rounds, r_{min} becomes *min* because it moves during $4 * id_{r_{min}} * n$ steps in the right direction. This function is maximal when $nb = 0$, therefore in at most $16 * id_{r_{min}} * n^2$ rounds r_{min} becomes *min* because it moves during $4 * id_{r_{min}} * n$ steps in the right direction. Now consider the case where r_{min} becomes *min* because it meets a robot that permits it to deduce that it is *min*. Once r_{min} is stuck more than $4 * n$ rounds after time t, we have seen that it becomes *min*. Since we consider the worst case such that r_{min} does not become *min* because it moves during $4 * id_{r_{min}} * n$ steps in the right direction, this implies that in at most $(4 * id_{r_{min}} * n - 1) * 4 * n$ rounds r_{min} becomes *min*. Therefore, whatever the situation, Phase `M` is bounded.

Moreover, by Case 1.1 and the previous paragraph, we can conclude that Phase `M` is in $O(id_{r_{min}} * n^2)$ rounds.

Now we consider Phase `K` of $\mathbb{GDG}$. In this phase r_{min} is *min* and waits for a $towerMinConfiguration$ to be formed. We take back the arguments used in the proofs of Lemmas 6.15 and 6.16 to prove that this phase is bounded.

Phase `K`: Case 2.1: There is a potentialMin in the execution.

For this case we take back the arguments of the proof of Lemma 6.15.

If before being min, r_{min} is a $righter$, then all the robots that are not located on node u are $potentialMin$, $dumbSearcher$, and $awareSearcher$. As long as it is not on node u, a $potentialMin$ either executes Rule $\mathbf{M_8}$, or it becomes an $awareSearcher$ (Rule $\mathbf{M_5}$). While executing Rule $\mathbf{M_8}$, a $potentialMin$ stays a $potentialMin$ and has the same behavior as if it was executing the function SEARCH(). Moreover, as long as they are not on node u, $dumbSearcher$ and $awareSearcher$ robots stay either $dumbSearcher$ or $awareSearcher$ and execute the function SEARCH(). Therefore, by definition of the function SEARCH() (refer to Phase M case 1.3 of this proof) and by Lemma 6.13, at most $3 * n$ rounds are needed (in $\mathcal{AC}$ rings) for a robot r such that $kind_r \in \{potentialMin, dumbSearcher, awareSearcher\}$ to be located on node u. Indeed, these $3 * n$ rounds are needed especially when a $potentialMin$, $dumbSearcher$ or $awareSearcher$ moves in one direction during n steps and then is stuck on the adjacent node of u, then n steps are needed for a robot of this kind to be also located on this node and thus to point to the opposite direction, then in at most n additional steps a robot of this kind is located on u. By Rule $\mathbf{K_3}$, this implies that at most $3 * n$ rounds are necessary for a supplementary $waitingWalker$ to be located on node u. Therefore, at most $(\mathcal{R}-3)*3*n$ rounds are needed for a $towerMinConfiguration$ to be formed.

Now consider the case where before being min, r_{min} is a $potentialMin$.

In this case among the robots that are not on node u, there are $dumbSearcher$, $awareSearcher$ and at most one $righter$.

For all the cases of Case 2.1 of the proof of Lemma 6.15, at most $(\mathcal{R}-4)*3*n$ rounds are needed for $\mathcal{R}-4$ $waitingWalker$ to be located on u (for the same reasons as the one explained in the previous paragraph). Then among the robots that are not on node u, it exists at most one $righter$, and 2 robots that are either $dumbSearcher$ or $awareSearcher$. At most n rounds are needed for the $righter$ to be stuck on the node called v in the proof of Lemma 6.15, and then at most n rounds are needed for a $dumbSearcher$ or an $awareSearcher$ to be also located on node v (and thus, by Rule $\mathbf{M_7}$, for all the robots that are not on node u to be either $dumbSearcher$ or $awareSearcher$), and then at most n additional rounds are needed for one of the robot to reach node u. Therefore, for all the cases of Case 2.1 of the proof of Lemma 6.15, at most $(\mathcal{R}-4)*3*n+3*n$ rounds are needed for a $towerMinConfiguration$ to be formed.

If we consider Case 2.2 of the proof of Lemma 6.15, similarly as in the previous case, at most $(\mathcal{R}-4)*3*n+3*n$ rounds are needed for Rule $\mathbf{Term_2}$ to be executed.

Case 2.2: There is no potentialMin in the execution.

For this case we take back the arguments of the proof of Lemma 6.16.

Just after r_{min} becomes min, it takes at most $n * n$ rounds for a robot r to join the node where r_{min} is located. Indeed, as long as no robot is on node u with r_{min}, as a $minWaitingWalker$, all the robots except r_{min} are $righter$. By the same arguments as the one used in Phase M Case 1.1 of this proof, a $righter$ cannot be stuck more than n rounds on the same node, otherwise Rule $\mathbf{M_6}$ is executed, which is a contradiction with the fact that there is no $potentialMin$. Moreover, a $righter$ can move from at most n steps in the right direction to reach u.

Once r is on node u an adjacent right edge to u is present in at most n rounds, otherwise Rule $\mathbf{Term_1}$ is executed. Therefore, once r is on node u, in at most n rounds it becomes an $awareSearcher$. From this time, either it is possible for all the $righter$ to become $awareSearcher$ or it exists at least one $righter$ that is stuck on node u. In the first case at most $2 * n$ rounds are needed for all the $righter$ to become $awareSearcher$ (either because an $awareSearcher$ meets them, or because they are located on u such that there is an adjacent right edge to u). By the arguments above, we know that if all the robots that are not located on node u are $awareSearcher$, and if there are more than 3 such robots, then in a most $3*n$ rounds one robot of this kind reaches

node u. Therefore, for $\mathcal{R}-3$ $waitingWalker$ to be located on node u, with r_{min}, at most $(\mathcal{R}-3)*3*n$ supplementary rounds are needed. In the second case, at most $2*n$ rounds are needed for some of the $righter$ to reach node u (and to be stuck on this node). Since the robots that are not on node u are $awareSearcher$ and since at least one $righter$ is stuck on node u, by the same arguments as above, at most $(\mathcal{R}-3)*3*n$ additional rounds are needed for Rule **Term₂** to be executed.

Therefore, in this case, at most $n*n+n+2*n+(\mathcal{R}-3)*3*n$ rounds are needed for either Phase K to be achieved or Rule **Term₂** to be executed.

Therefore Phase K is bounded. Moreover, by the two previous cases, we can conclude that Phase K is in $O(\mathcal{R}*n+n^2)$ rounds.

Now we consider Phase W of $\mathbb{GDG}$. In this purpose we take back the arguments used in the proof of Lemma 6.21.

Phase W: Here we consider the worst execution in terms of times. Therefore, we consider that Rules **Term₁** and **Term₂** are executing at the very last moment. The robots r_1 and r_2 that are not involved in T at time t_{tower} are such that $kind_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $kind_{r_2} \in \{dumbSearcher, awareSearcher\}$. Therefore, as explained previously, each time the $headWalker$, or the $minTailWalker$ / $tailWalker$ robots move from one step in the right direction, they can be stuck at most during $3*n$ rounds; otherwise either Rule **Term₁** or Rule **Term₂** is executed. Indeed, this is especially the case when the $headWalker$ and the $minTailWalker$ / $tailWalker$ are stuck on the same node. In fact, it takes at most n rounds for r_1 to be stuck on the other extremity the missing edge. At most n supplementary rounds are needed for r_2 to reach the node where r_1 is stuck (and therefore for one robot to change its direction), and then n other rounds are needed for one of these robots to reach the node where the $headWalker$ and the $minTailWalker$ / $tailWalker$ are stuck (and thus for Rule **Term₂** to be executed). Therefore, Phase W is achieved in at most $2*n*(3*n)$ rounds since the $headWalker$ and the $minTailWalker$ / $tailWalker$ robots have to move alternatively during n steps to complete Phase W. In other words, Phase W is bounded and is in $O(n^2)$ rounds.

Now we consider Phase T of $\mathbb{GDG}$. In this purpose we take back the arguments used in the proof of Lemma 6.21.

phase T: Using similar arguments as the one used in Phase W, once the $headWalker$ and the $minTailWalker$ / $tailWalker$ stop to move forever, if they are located on a same node, at most $3*n$ rounds are necessary for Rule **Term₂** to be executed. In the case where the $headWalker$ and the $minTailWalker$ / $tailWalker$ stop to move forever, if they are located on different nodes, at most $2*n+2*n$ rounds are necessary for Rule **Term₂** to be executed. Indeed, at most $2*n$ rounds are necessary for each of the two robots that are not involved in T at time t_{tower} to be located on the node where the $minTailWalker$ /$tailWalker$ is located. This is true whatever the interactions between r_1 and r_2 and whatever the interactions between r_1 (resp. r_2) and the $headWalker$ since in an $\mathcal{AC}$ ring there is at most one edge missing at each instant time (and in this precise case the missing edge is between the node where the $headWalker$ is located and the node where the $minTailWalker$ / $tailWalker$ are located). Therefore, Phase T is bounded and is in $O(n)$ rounds.

In conclusion each of the four phases of algorithm $\mathbb{GDG}$ are bounded when executed in an $\mathcal{AC}$ ring, therefore $\mathbb{GDG}$ solves $\mathbb{G}_W$ in $\mathcal{AC}$ rings. Moreover, $\mathbb{GDG}$ solves $\mathbb{G}_W$ in $\mathcal{AC}$ rings in $O(id_{r_{min}}*n^2+\mathcal{R}*n)$ rounds. □

Now, we consider the case of $\mathcal{BRE}$ rings. We prove, in Theorem 6.5, that $\mathbb{GDG}$ satisfies $\mathbb{G}$ in $\mathcal{BRE}$ rings. To prove this, we first need to prove the following lemma that it useful to bound Phase K of $\mathbb{GDG}$ in $\mathcal{BRE}$ rings. We prove the following lemma using the arguments of the proof

of Lemma 6.14 and the fact that in a $\mathcal{BRE}$ ring each edge appears at least once every δ units of time.

Lemma 6.22. *If the ring is a $\mathcal{BRE}$ ring and if there is no $towerMinConfiguration$ in the execution but there exists at a time t at least 3 robots such that they are either $potentialMin$, $dumbSearcher$ or $awareSearcher$, then at least a $potentialMin$, a $dumbSearcher$ or an $awareSearcher$ reaches the node u between time t and time $t + n * \delta$ included, with $\delta \geq 1$.*

Theorem 6.5. *$\mathbb{GDG}$ satisfies $\mathbb{G}$ in $\mathcal{BRE}$ rings under $\mathcal{M}_{gathering}$ in $O(n*\delta*(id_{r_{min}}+\mathcal{R}))$ rounds.*

Proof. By Lemma 6.3, $\mathbb{GDG}$ solves $\mathbb{G}_E$ in $\mathcal{RE}$ rings. Therefore, since $\mathcal{BRE} \subset \mathcal{RE}$, then $\mathbb{GDG}$ also solves $\mathbb{G}_E$ in $\mathcal{BRE}$ rings. We want to prove that $\mathbb{GDG}$ solves $\mathbb{G}$ in $\mathcal{BRE}$ rings. Therefore, we have to prove that each phase of the algorithm is bounded.

Phase `M`: By Corollary 6.4, we know that only r_{min} becomes min in finite time. By Lemma 6.9, before being min, r_{min} is either a $righter$ or a $potentialMin$ robot. By Lemma 6.2, if, at a time t, a robot is a $righter$ or a $potentialMin$ robot, then it points to the $right$ direction from the beginning of the execution until the Look phase of time t. Since initially all the robots are $righter$, and since, by the rules of $\mathbb{GDG}$, only $righter$ can become $potentialMin$ (refer to Rule $\mathbf{M_6}$), then by Observations 6.2 and 6.1, a robot that is a $righter$ (resp. $potentialMin$) is a $righter$ (resp. is either a $righter$ or a $potentialMin$) since the beginning of the execution. Besides, each time r_{min}, as a $righter$ or as a $potentialMin$, crosses an edge in the right direction, it increases its variable $rightSteps$ of one (refer to Rules $\mathbf{M_8}$ and $\mathbf{M_6}$). Therefore, since each edge of the footprint of a $\mathcal{BRE}$ ring is present at least once every δ units of time, by definition of min and of the predicate $MinDiscovery()$ of Rule $\mathbf{M_1}$, r_{min} becomes min in at most $4 * n * id_{r_{min}} * \delta$ rounds. Hence, Phase `M` is bounded and is in $O(id_{r_{min}} * n * \delta)$.

Phase `K`: Now, consider the execution when r_{min} just becomes min. Therefore, we consider the execution once r_{min} is $minWaitingWalker$. By Corollary 6.5, we know that in finite time a $towerMinConfiguration$ is formed. By Lemma 6.6, there is only one $towerMinConfiguration$ in the whole execution. Therefore, before a $towerMinConfiguration$ is formed, by the rules of $\mathbb{GDG}$ and since initially all the robots are $righter$, there are only $righter$, $potentialMin$, $dumbSearcher$, $awareSearcher$, $minWaitingWalker$ and $waitingWalker$ robots. By Lemma 6.7, we know that all the $minWaitingWalker$ and $waitingWalker$ robots are located on a same node and do not move. By Rule $\mathbf{K_3}$, if a $potentialMin$, a $dumbSearcher$ or an $awareSearcher$ is located on the same node as a $minWaitingWalker$, it becomes $waitingWalker$ $(*)$. If there is no more $righter$ robot in the execution, we use Lemma 6.22 and $(*)$ multiple times to prove that it takes at most $n * \delta * (\mathcal{R} - 3)$ rounds for a $towerMinConfiguration$ to be formed. To prove that Phase `K` is bounded, we hence have to prove that the number of rounds that are necessary to stop to have $righter$ in the execution is bounded.

If a $righter$ is located on the same node as the $minWaitingWalker$ while there is an adjacent right edge to its location, then by Rule $\mathbf{K_4}$, the $righter$ becomes an $awareSearcher$ and moves on the right. If a $righter$ is located only with $\mathcal{R} - 2$ other $righter$, they all execute Rule $\mathbf{M_6}$, hence one becomes $potentialMin$ while the others become $dumbSearcher$. If a $righter$ is located either with a $dumbSearcher$ or with an $awareSearcher$, then it becomes an $awareSearcher$ (Rule $\mathbf{M_7}$). Note that, by Lemma 6.10, since we consider the execution once r_{min} is min, it cannot exist a $righter$ and a $potentialMin$ in the execution. Therefore, a $righter$ cannot meet a $potentialMin$. In all the other cases, (a $righter$ that is isolated, a $righter$ that is only with others $righter$ on its node such that $|NodeMate()| < \mathcal{R}-2$, and a $righter$ that is located on the same node as the $minWaitingWalker$ while there is no adjacent right edge to its location) a $righter$ stays a $righter$ and

points to the *right* direction (Rule $\mathbf{M_8}$). Therefore, by Observation 6.2 and since each edge of the footprint of a $\mathcal{BRE}$ ring is present at least once every δ units of time, it takes at most $n * \delta$ rounds in order to stop having *righter* robots in the execution. Indeed, even if a *righter* does not execute Rule $\mathbf{M_7}$ or Rule $\mathbf{M_6}$, at most $n * \delta$ rounds are needed for it to be located on the node where the $minWaitingWalker$ is located while there is an adjacent right edge to its position. Hence, Phase `K` is bounded and is in $O(n * \mathcal{R} * \delta)$.

Once a $towerMinConfiguration$ is present in the execution, the robots forming this $towerMinConfiguration$ execute Rule $\mathbf{K_1}$. While executing this rule, the robot r with the maximum identifier among the robots involved in this $towerMinConfiguration$ becomes $headWalker$ while the $minWaitingWalker$ becomes $minTailWalker$ and the other robots involved in this $towerMinConfiguration$ become $tailWalker$. Note that, by Corollary 6.4, only r_{min} can be min, and therefore, since r_{min} is the robot with the minimum identifier among all the robots of the system and since at least 2 robots are involved in a $towerMinConfiguration$, r_{min} cannot become $headWalker$. By Lemma 6.6 and by the rules of $\mathbb{GDG}$, only r can be $headWalker$ during the execution.

There is no rule in $\mathbb{GDG}$ permitting a $tailWalker$ or a $minTailWalker$ robot to become another kind of robot. A $headWalker$ can become a $leftWalker$. Let then consider the two following cases.

Case 1: r is a *headWalker* during the whole execution.

Phase `W`: A $headWalker$ can execute Rules $\mathbf{T_2}$, $\mathbf{T_3}$ and $\mathbf{W_1}$. Since r does not become a $leftWalker$, it cannot execute Rule $\mathbf{T_2}$. Moreover, since we consider the worst-case execution in terms of time, this implies that r is able to execute Rule $\mathbf{W_1}$ entirely. This means that r is able to execute Rule $\mathbf{W_1}$ until its variable *walkSteps* reaches the value n. In other words, r is able to execute Rule $\mathbf{W_1}$ until it executes Rule $\mathbf{T_3}$.

In this case, the $tailWalker$ and $minTailWalker$ are also able to execute Rule $\mathbf{W_1}$ entirely. Indeed, if, at a time t', while executing Rule $\mathbf{W_1}$ or Rule $\mathbf{T_3}$, the $headWalker$ is waiting on its node for the $tailWalker$ and the $minTailWalker$ to join it while there is an adjacent left edge to its position, and if at time $t' + 1$ the $tailWalker$ and the $minTailWalker$ have not joined it on its node, this necessarily implies that they stop their execution, otherwise by Rule $\mathbf{W_1}$ they would have joined it. Moreover, if such an event happens, r executes Rule $\mathbf{T_1}$ and therefore becomes a $leftWalker$, which leads to a contradiction.

If the $headWalker$ and the $minTailWalker/tailWalker$ execute Rule $\mathbf{W_1}$ entirely, this implies that they move alternatively in the right direction during n steps. Since each edge of the footprint of a $\mathcal{BRE}$ ring is present at least once every δ units of time, this takes at most $2 * n * \delta$ rounds. Phase `W` being composed only of the execution of Rule $\mathbf{W_1}$, this phase is bounded.

Phase `T`: Call t_v the time at which the $headWalker$ and $minTailWalker/tailWalker$ robots finish to execute Rule $\mathbf{W_1}$ entirely. Since the $headWalker$ and $minTailWalker/tailWalker$ start the execution of Rule $\mathbf{W_1}$ on the same node, at time t_v, they are on the same node v.

Call r_1 and r_2 the two robots that are not involved in the $towerMinConfiguration$ at time t_{tower}.

If at time t_v, r_1 and r_2 are on node v, then Rule $\mathbf{Term_1}$ is executed at time t_v. In this case, by Lemma 6.18, Phase `T` last 0 round, hence it is bounded.

If at time t_v, only one robot among r_1 and r_2 is located on node v, then Rule $\mathbf{Term_2}$ is executed at time t_v. Hence, by Lemma 6.18, $\mathcal{R} - 2$ robots are terminated on node v at

time t_v. By Lemma 6.17, at time t_{tower}, r_1 and r_2 are such that $kind_{r_1} \in \{righter, potentialMin, awareSearcher, dumbSearcher\}$ and $kind_{r_2} \in \{awareSearcher, dumbSearcher\}$. By the movements of the robots given in the proof of Lemma 6.21, and since each edge of the footprint of a $\mathcal{BRE}$ ring is present at least once every δ units of time, it takes at most $n * \delta$ rounds for the last robot to reach node v. Therefore, it takes at most $n * \delta$ rounds for Rule $\mathbf{Term_1}$ to be executed, and thus, by Lemma 6.18, for all the robots to be terminated on node v. Hence, in this case Phase `T` last at most $n * \delta$ rounds, therefore it is bounded.

Now, consider that at time t_v neither r_1 nor r_2 is located on node v. In this case, at time t_v, the $headWalker$ and $minTailWalker/tailWalker$ execute Rule $\mathbf{T_3}$. While executing Rule $\mathbf{T_3}$, the $headWalker$ (resp. $minTailWalker/tailWalker$) stays a $headWalker$ (resp. $minTailWalker/tailWalker$) and points to the $\perp$ direction. Then, by the rules of $\mathbb{GDG}$, they can only execute Rule $\mathbf{T_3}$ until they terminate. Therefore, they remain on node v from time t_v until the end of their execution. Moreover, as noted previously, by Lemma 6.17, at time t_{tower}, r_1 and r_2 are such that $kind_{r_1} \in \{righter, potentialMin, awareSearcher, dumbSearcher\}$ and $kind_{r_2} \in \{awareSearcher, dumbSearcher\}$. By the movements of the robots given in the proof of Lemma 6.21, since each edge of the footprint of a $\mathcal{BRE}$ ring is present at least once every δ units of time, it takes at most $2 * n * \delta$ rounds for r_1 and r_2 to both reach the node v (in case r_1 and r_2 meet on an adjacent node of v after at most $n * \delta$ rounds of movements in the same direction). In the case the two robots reach node v at the same time, then Rule $\mathbf{Term_1}$ is executed, hence, by Lemma 6.18, all the robots terminate at that time. In the case the two robots do not reach node v at the same time, then the first one that reaches v permits the execution of Rule $\mathbf{Term_2}$ (hence, by Lemma 6.18, permits the termination of $\mathcal{R} - 2$ robots on node v) and the second that reaches v permits the execution of Rule $\mathbf{Term_1}$. Hence, Phase `T` last at most $2 * n * \delta$ rounds, therefore it is bounded.

Case 2: It exists a time at which r is a *leftWalker*.

By the explanations given in the Case 1, Phase `W`, at most $2 * n * \delta$ rounds are needed for r to become $leftWalker$ and for the $\mathcal{R} - 2$ other robots to terminate their execution on a node v.

By the rules of $\mathbb{GDG}$, a $leftWalker$ only executes Rule $\mathbf{T_1}$. While executing this rule, a robot points to the $left$ direction and stays a $leftWalker$. Since each edge of the footprint of a $\mathcal{BRE}$ ring is present at least once every δ units of time, such a robot reaches the node v in at most $n * \delta$ rounds. Hence, in this case, Phases `W` and `T` take at most $3 * n * \delta$ rounds, hence they are bounded.

By the two previous cases, phase `W` and Phase `T` take $O(n * \delta)$ rounds.

Whatever the $\mathcal{BRE}$ ring considered, each phase of $\mathbb{GDG}$ is bounded, therefore, $\mathbb{GDG}$ solves $\mathbb{G}$ in $\mathcal{BRE}$ rings. Moreover, $\mathbb{GDG}$ solves $\mathbb{G}$ in $\mathcal{BRE}$ rings in $O(n * \delta * (id_{r_{min}} + \mathcal{R}))$ rounds. □

Now, we consider the case of $\mathcal{ST}$ rings. We know that $\mathcal{ST}$ rings are $\mathcal{BRE}$ rings such that $\delta = 1$, hence, by Theorem 6.5, we can deduce the following corollary.

Corollary 6.6. *$\mathbb{GDG}$ satisfies $\mathbb{G}$ in $\mathcal{ST}$ rings under $\mathcal{M}_{gathering}$ in $O(n * (id_{r_{min}} + \mathcal{R}))$ rounds.*

From Theorems 6.2, 6.3, 6.4, 6.5, and Corollary 6.6, we can conclude the following theorem.

Theorem 6.6. *$\mathbb{GDG}$ is a gracefully degrading algorithm with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$, for the gathering problem under $\mathcal{M}_{gathering}$.*

6.4 Summary

In this chapter, we apply for the first time (in an explicit way) the gracefully degrading approach to robot networks. This approach consists in circumventing impossibility results in highly dynamic systems by providing algorithms that adapt themselves to the dynamics of the graphs: they solve the problem under weak classes of dynamic graphs (where the connectivity assumptions are strong) and only guarantee that some weaker but related problems are satisfied whenever the dynamics increase and make the original problem impossible to solve.

Focusing on the classical problem of gathering a squad of autonomous robots, we introduce a set of weaker variants of this problem that preserve its safety (in the spirit of the indulgent approach that shares the same underlying idea). Indeed, we define $\mathbb{G}_E$ (where all the robots terminate their execution on the same node of the graph in finite time), $\mathbb{G}_W$ (in which all robots but (at most) one terminate their execution on the same node of the graph in finite and bounded time) and $\mathbb{G}_{EW}$ (where all robots but (at most) one terminate their execution on the same node of the graph in finite time) that derive from $\mathbb{G}$ (the original version of the gathering problem in which all the robots terminate their execution on the same node of the graph in finite and bounded time). Motivated by a set of impossibility results, we propose a gracefully degrading gathering algorithm solving $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings, $\mathbb{G}_W$ in $\mathcal{AC}$ rings, $\mathbb{G}_E$ in $\mathcal{RE}$ rings and $\mathbb{G}$ in $\mathcal{BRE}$ and $\mathcal{ST}$ rings (refer to Table 6.1 for a summary of our results).

Chapter 7

Conclusion on Graceful Degradation

Gracefully degrading algorithms may be executed in multiple dynamic environments and in environments in which the dynamics change with time. They are used to circumvent impossibility results in dynamic systems: they solve the problem studied in some dynamic environments in which it is solvable and degrade it when the dynamics of the environments increase and make it impossible. The goal of this part was to extend the gracefully degrading approach to robot networks evolving in dynamic graphs. On this purpose, we considered the gathering problem in dynamic rings.

In this part, we first have presented, in Chapter 5, the state of the art about the gracefully degrading approach as well as the state of the art concerning the gathering problem in dynamic graphs. The gathering problem being impossible in $\mathcal{AC}$ rings [122] (also refer to Section 6.1), it is a good case study for the conception of a gracefully degrading algorithm. However, none of the gathering (or rendezvous) algorithms in dynamic graphs of the state of the art is explicitly gracefully degrading. Moreover, even when being (without telling it) gracefully degrading, none of the gathering (or rendezvous) algorithms of the state of the art is gracefully degrading with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$, preserves the safety of the gathering problem, and imposes that a non-zero number of robots terminate their execution.

Therefore, to fill the lack of the state of the art, we have presented, in Chapter 6, the first gracefully degrading gathering algorithm with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$, that preserves the safety of the gathering problem, and that guarantees that at least $\mathcal{R} - 1$ robots terminate their execution. The gracefully degrading algorithm we provide solves $\mathbb{G}$ in $\mathcal{ST}$ and $\mathcal{BRE}$ rings, $\mathbb{G}_E$ in $\mathcal{RE}$ rings, $\mathbb{G}_W$ in $\mathcal{AC}$ rings and $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings (refer to Section 5.3 for the definitions of $\mathbb{G}$, $\mathbb{G}_E$, $\mathbb{G}_W$ and $\mathbb{G}_{EW}$).

Short-term perspectives. The algorithm we proposed makes multiple assumptions on the robots (fully-synchronous, communication when forming a tower, strong multiplicity detection, identifiers, chirality, persistent memory, knowledge of the total number of robots $\mathcal{R}$ and of the size of the ring n). These assumptions are necessary for our algorithm. However, it could be interesting to study the necessity of these assumptions in order to conceive a gracefully degrading algorithm for the gathering problem with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$. In particular, it could be interesting to consider another computation model (i.e., consider the semi-synchronous model or the asynchronous model).

Moreover, our algorithm is adapted only for ring-shaped graphs. Hence, it would be necessary to lead a similar study as ours but considering other topologies of graphs. The classical approach is first to generalize solutions to grids and then to tori as intermediate steps, before considering any graph.

The gathering problem is certainly not the only problem using robots that becomes impossible when the dynamics of the graphs increase. For instance, the scattering problem with robots endowed with local vision seems to be impossible in $\mathcal{AC}$ or $\mathcal{COT}$ graphs. Therefore, it would be nice to find all the problems that become impossible when the dynamics of the graphs increase, and for each of these problems, find weaker versions of them in order to conceive gracefully degrading algorithms to solve them.

Midterm perspectives. In addition to be gracefully degrading, one may observe that our algorithm is optimal for the specifications of the gathering problem and classes of dynamic graphs we consider: our algorithm solves the best version of the gathering problem (among the one we propose) in the classes of dynamic graphs we consider (refer to Table 6.1). Note that this does not mean that there is no other possible variant of the gathering problem or no other gracefully degrading gathering algorithm that could be considered as better. We can generalize this intuition with the following notion:

Definition 7.1 (Optimal gracefully degrading algorithm)**.** *Given a model $\mathcal{M}$, a problem $\mathcal{P}_0$, a set $\mathcal{S}_\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$ of problems either equal to or weaker than $\mathcal{P}_0$, with $|\mathcal{S}_\mathcal{P}| \geq 1$, and a set $\mathcal{S}_\mathcal{C} = \{\mathcal{C}_0, \ldots, \mathcal{C}_k\}$ of classes of dynamic graphs such that:*

1. *$|\mathcal{S}_\mathcal{C}| \geq 2$.*
2. *For all i, with $1 \leq i \leq k, \mathcal{C}_0 \subset \mathcal{C}_i$.*
3. *For all i, with $0 \leq i \leq k$, and for all j, with $0 \leq j \leq k$, if $\mathcal{C}_i \subset \mathcal{C}_j$ then $\mathcal{P}_i$ is either stronger than or equal to the problem $\mathcal{P}_j$ (i.e., $\mathcal{SE}_{\mathcal{P}_i} \subseteq \mathcal{SE}_{\mathcal{P}_j}$).*
4. *There exists at least one i, with $1 \leq i \leq k$, such that $\mathcal{P}_0$ is impossible to solve in $\mathcal{C}_i$ under $\mathcal{M}$.*

a $(\mathcal{P}_0, \mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{C})$-optimal gracefully degrading algorithm $\mathcal{A}$ for $\mathcal{P}_0$ under $\mathcal{M}$ satisfies:

- *$\mathcal{A}$ is a $(\mathcal{P}_0, \mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{C})$-gracefully degrading algorithm for $\mathcal{P}_0$ under $\mathcal{M}$.*
- *For all i, with $0 \leq i \leq k$, and for all j, with $0 \leq j \leq k$, if $\mathcal{C}_i \subset \mathcal{C}_j$ and $\mathcal{A}$ does not solve $\mathcal{P}_i$ in $\mathcal{C}_j$ under $\mathcal{M}$ then $\mathcal{P}_i$ is impossible to solve in $\mathcal{C}_j$ under $\mathcal{M}$.*

We say that an algorithm is optimal gracefully degrading with respect to a family of classes of dynamic graphs $\mathcal{S}_\mathcal{C}$, for a problem $\mathcal{P}_0$, and a set of weaker problems $\mathcal{S}_\mathcal{P}$, if this algorithm is $(\mathcal{P}_0, \mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{C})$-optimal gracefully degrading.

Our algorithm is then an optimal gracefully degrading gathering algorithm with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$, for $\mathbb{G}$, $\mathbb{G}_E$, $\mathbb{G}_W$, and $\mathbb{G}_{EW}$.

Let us analyze below which of the gathering algorithms of the state of the art that were gracefully degrading with respect to some classes of dynamic graphs are also optimal gracefully degrading with respect to some classes of dynamic graphs for some versions of the gathering problem. Depending on the classes of dynamic graphs and the weaker versions of the gathering problem considered, a gathering algorithm could be or not optimal gracefully degrading. In the following, we consider the classes of dynamic graphs (among the one we studied in this thesis) we think the authors (of the algorithms of the state of the art) would have considered, and the weaker versions of the gathering problem we think they would have focused on. Recall that $\mathcal{ST} \subset \mathcal{BRE} \subset \mathcal{RE} \subset \mathcal{COT}$ and $\mathcal{ST} \subset \mathcal{AC} \subset \mathcal{COT}$.

Algorithms proposed by Izumi et al. [110]: Their algorithms are optimal gracefully degrading (in the case where the robots do not start from a periodic configuration) with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$, for $\mathbb{G}$, $\mathbb{G}_E$, and a weaker version $\mathcal{V}$ of the gathering problem where the robots must end up (without necessarily terminating) on at most two different nodes (not necessarily adjacent). Indeed, their algorithms solve $\mathbb{G}$ in $\mathcal{ST}$ and $\mathcal{BRE}$ rings, $\mathbb{G}_E$ in $\mathcal{RE}$ rings, and $\mathcal{V}$ in $\mathcal{AC}$ and $\mathcal{COT}$ rings (and $\mathbb{G}$ is impossible to solve in $\mathcal{RE}$ rings, and $\mathbb{G}$ and $\mathbb{G}_E$ are impossible to solve in $\mathcal{AC}$ and $\mathcal{COT}$ rings).

Algorithms proposed by Di Luna et al. [122]: When the initial configuration of the robots is not periodic, their algorithms (except the one considering robots endowed with the cross detection and without chirality) are optimal gracefully degrading with respect to $\mathcal{ST}$ and

$\mathcal{AC}$, for $\mathbb{G}$, and the near-gathering with termination. Indeed, their algorithms solve $\mathbb{G}$ in $\mathcal{ST}$ rings, and the near-gathering with termination in $\mathcal{AC}$ rings (and $\mathbb{G}$ is impossible to solve in $\mathcal{AC}$ rings).

Algorithm proposed by Ooshita and Datta [128]: Their algorithm is optimal gracefully degrading with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$, for $\mathbb{G}$, $\mathbb{G}_E$, and the near-gathering without termination. Indeed, it solves $\mathbb{G}$ in $\mathcal{ST}$ and $\mathcal{BRE}$ rings, $\mathbb{G}_E$ in $\mathcal{RE}$ rings, and the near-gathering without termination in $\mathcal{AC}$ and $\mathcal{COT}$ rings (and $\mathbb{G}$ is impossible to solve in $\mathcal{RE}$ rings, and $\mathbb{G}$ and $\mathbb{G}_E$ are impossible to solve in $\mathcal{AC}$ and $\mathcal{COT}$ rings).

We proved that our algorithm provides for each class of dynamic graphs studied the best specification of the gathering problem among the ones we considered (refer to Table 6.1). We do not claim that another algorithm could not be able to satisfy stronger specifications among the infinity of variants one can propose of the original gathering specification. For instance, one may consider a specification of the gathering where all the robots terminate their execution on the same node in finite time and such that at least $x > 0$ of these robots terminate in bounded time. This specification is stronger than $\mathbb{G}_E$. Perhaps another gracefully degrading gathering algorithm would be able to solve this new specification instead of $\mathbb{G}_E$ in classes of dynamic graphs in which our algorithm solves $\mathbb{G}_E$. If, for each class of dynamic graphs considered, this gracefully degrading gathering algorithm satisfies stronger (or equal) specifications of the gathering problem compared to the one we solve, then it could be considered stronger than our algorithm.

Therefore, one question that comes in mind is "Is there a strongest gracefully degrading gathering algorithm?" We call a gracefully degrading algorithm an optimum gracefully degrading algorithm if it solves the best versions of the gathering problem in the classes of dynamic graphs analyzed. Note that, if such an optimum gracefully degrading algorithm exists, it is necessarily optimal gracefully degrading. Moreover, note that, since some problems are not comparable, it could exist multiple optimum gracefully degrading algorithms for a same problem.

Among the articles of the state of the art that are optimal gracefully degrading, we detail below which of them are optimum gracefully degrading. For this study we still consider the weaker versions of the gathering problem and classes of dynamic graphs we think the authors would have focused on.

Algorithms proposed by Izumi et al. [110]: Even if their algorithms are optimal gracefully degrading with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$, for $\mathbb{G}$, $\mathbb{G}_E$, and a weaker version of the gathering problem where the robots must end up (without necessarily terminating) on at most two different nodes (not necessarily adjacent), they are not optimum gracefully degrading. Indeed, in $\mathcal{COT}$ rings, it is possible to solve the near-gathering without termination (refer to the algorithm of Ooshita and Datta [128]). Moreover, in $\mathcal{AC}$ rings, it is possible to solve the near-gathering with termination (refer to the algorithms of Di Luna et al. [122]). Hence, in $\mathcal{COT}$ and $\mathcal{AC}$ rings, we are sure that their algorithms do not satisfy the best versions possible of the gathering problem, proving that their algorithms are not optimum gracefully degrading.

Algorithms proposed by Di Luna et al. [122]: Their algorithms are optimal gracefully degrading with respect to $\mathcal{ST}$ and $\mathcal{AC}$, for $\mathbb{G}$, and the near-gathering with termination. Nothing indicates that their algorithms are not optimum gracefully degrading. Hence, there is an open question on this subject.

Algorithm proposed by Ooshita and Datta [128]: Even if their algorithm is optimal gracefully degrading with respect to $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$, for $\mathbb{G}$, $\mathbb{G}_E$, and the near-gathering without termination, it is not optimum gracefully degrading. In $\mathcal{AC}$ rings, their robots may end up on two adjacent nodes without terminating their execution. However, in $\mathcal{AC}$ rings, it is possible to solve the near-gathering with termination (refer to the algorithms

of Di Luna et al. [122]). Hence, in $\mathcal{AC}$ rings, we are sure that their algorithm does not satisfy the best version possible of the gathering problem, proving that their algorithm is not optimum gracefully degrading.

The weaker version of the gathering problem solved by Di Luna et al. [122] in $\mathcal{AC}$ rings and the weaker version of the gathering problem we solved in $\mathcal{AC}$ rings are not comparable. Hence, nothing indicates that our algorithm is not optimum gracefully degrading. An interesting open question is to analyze whether our algorithm is an optimum gracefully degrading algorithm.

Aside gathering in robot networks, defining formally a general form of optimum degradation in the gracefully degrading approach seems to be a challenging future work.

When we study gracefully degrading algorithms, multiple classes of dynamic graphs are considered. In this thesis we consider $\mathcal{ST}$, $\mathcal{BRE}$, $\mathcal{RE}$, $\mathcal{AC}$, and $\mathcal{COT}$ graphs. However, multiple other classes of dynamic graphs exist (refer to the hierarchy of classes of dynamic graphs presented in Section 2.1.2). Hence, another interesting perspective is to determine the classes of dynamic graphs that are important to study to apply a gracefully degrading approach to a problem. It is not necessarily important to study every class of dynamic graphs. Indeed, considering all the possible classes of dynamic graphs would make the gracefully degrading algorithm very complex. Moreover, some classes of dynamic graphs presented in a theoretical point of view are in practice not likely to occur. Therefore, it is possible not to handle these classes of dynamic graphs while conceiving a gracefully degrading algorithm. In other words, it is important to determine which classes of dynamic graphs to study depending on the trade-off between the probability of their occurrence and the complexity of the algorithms.

Part III

Speculation

Chapter 8

Introduction

Contents

In this part, we focus on speculative algorithms. Intuitively, a speculative algorithm is an algorithm that must satisfy the specification of the problem studied in any execution in which it is susceptible to be executed and that must be very efficient for the executions that are more likely to happen. The conception of a speculative algorithm for a given problem is motivated by high lower bounds obtained when considering some environments. We focus on the exploration problem (refer to Section 3.1.4 for the definitions of the different variants of the exploration problem). More precisely, the speculative algorithms we provide solve the perpetual exploration problem in which all the nodes of the graph must be visited infinitely often by at least one robot.

In this chapter, we first present, in Section 8.1, the state of the art about speculation, then, in Section 8.2, we present the state of the art about exploration in dynamic graphs. Finally, in Section 8.3, we give an overview of the next chapters of this part.

8.1 Speculation

8.1.1 Definition

Generally, while conceiving distributed algorithms the worst case is considered: the algorithms are optimized for this case. However, very often the worst case is not the most frequent one. For instance, generally, while conceiving distributed algorithms above internet the communications are assumed to be asynchronous. However, in practice internet is synchronous. Therefore, the lower bounds obtained (in the worst case) to solve problems do not necessarily fit the lower bounds obtained in the most frequent environments. Hence, it would be interesting to optimize algorithms for the most frequent case to circumvent the high lower bounds obtained only in some rare environments.

A speculative algorithm [114, 77, 9] is an algorithm that must satisfy its requirements (i.e., the specification of the problem considered) in any execution in which it is susceptible to be executed but also that must be very efficient for the executions that are more likely to happen.

In this thesis, we consider dynamic graphs. Depending on the context, some dynamic graphs are more frequent than others. Hence, in this thesis, we extend this notion of speculation to algorithms executed in dynamic graphs. A speculative algorithm for dynamic graphs must satisfy the specification of the problem considered in any class of dynamic graphs in which it is susceptible to be executed and must be optimized (for instance in terms of the time execution) for the classes of dynamic graphs in which it is more likely to be executed.

As indicated in Section 2.1.2, the strongest class of dynamic graphs of the classification of Casteigts et al. [40, 38] having a recurrent temporal connectivity is the class of $\mathcal{COT}$ graphs. This class includes all the other classes of dynamic graphs having a recurrent temporal connectivity. As an example, if we consider that an algorithm $\mathcal{A}$ (that has no knowledge on the dynamics of the graphs and must solve a task starting from any time) may be executed in any dynamic class of graphs, then to be speculative for dynamic graphs, it must be able to satisfy the specification of the problem studied in $\mathcal{COT}$ graphs. However, depending on the context of the applications, $\mathcal{COT}$ graphs are not necessarily the class of dynamic graphs the most common. Indeed, even if in dynamic systems there are unstable moments where the systems are very dynamic, some dynamic graphs are $\mathcal{ST}$ graphs, or at least they evolve in such a low way that they are static in the point of view of the applications that stop to be executed before changes in the topologies occur (i.e., the stable moments are long enough for the applications to stop being executed). Therefore, if $\mathcal{A}$ is executed in such an environment, to be a speculative algorithm for dynamic graphs, it could be optimized (for instance in terms of the time execution) for $\mathcal{ST}$ graphs.

To define formally what a speculative algorithm for dynamic graphs is, we first need to define the property which has to be optimized by such an algorithm. Given an algorithm $\mathcal{A}$ solving a problem $\mathcal{P}$ in a class of dynamic graphs $\mathcal{C}$ under a model $\mathcal{M}$, the property that $\mathcal{A}$ has to optimize is the measurement of performance $m_{\mathcal{A}}$ of $\mathcal{A}$: it is a function that associates to any execution of $\mathcal{A}$ in $\mathcal{C}$ under $\mathcal{M}$ its cost. Depending on the context, this measurement of performance may return a cost in terms of time, movements of the robots, memory, ...

In the following, by abuse of language, we denote by $f|_{\mathcal{C}}$ the restriction of the function f on all the executions in all the dynamic graphs that belong to $\mathcal{C}$.

We say that a problem $\mathcal{P}$ is not solvable in $\Theta(f)$ if there is no algorithm that satisfies $\mathcal{P}$ with a measurement of performance in $\Theta(f)$.

A speculative algorithm where the speculation is done on the dynamics of the system can be defined formally as follows:

Definition 8.1 (Speculative algorithm for dynamic graphs)**.** *Given a model $\mathcal{M}$, a problem $\mathcal{P}$, a set $\mathcal{F} = \{f_0, \ldots, f_k\}$ of functions, with $|\mathcal{F}| \geq 2$, a set $\mathcal{S}_{\mathcal{C}} = \{\mathcal{C}_0, \ldots, \mathcal{C}_k\}$ of classes of dynamic graphs such that:*

1. *$|\mathcal{S}_{\mathcal{C}}| \geq 2$.*
2. *For all i, with $0 \leq i \leq k-1, \mathcal{C}_i \subset \mathcal{C}_k$.*
3. *For all i, with $0 \leq i \leq k$, if $\mathcal{C}_i \subset \mathcal{C}_j$ then $f_i|_{\mathcal{C}_i} \in O(f_j|_{\mathcal{C}_j})$.*
4. *There exists at least one i, with $0 \leq i \leq k-1$, such that $\mathcal{P}$ is not solvable in $\Theta(f_i|_{\mathcal{C}_i})$ in $\mathcal{C}_k$ (implying that $f_i|_{\mathcal{C}_i} \in o(f_k|_{\mathcal{C}_k})$).*

a $(\mathcal{P}, \mathcal{F}, \mathcal{S}_{\mathcal{C}})$-speculative algorithm $\mathcal{A}$ for $\mathcal{P}$ under $\mathcal{M}$ (for dynamic graphs) is such that:

- *$\mathcal{A}$ satisfies the specification of $\mathcal{P}$ in $\mathcal{C}_k$ under $\mathcal{M}$.*
- *For all i, with $0 \leq i \leq k$, the measurement of performance $m_{\mathcal{A}}$ of $\mathcal{A}$ under $\mathcal{M}$ satisfies $m_{\mathcal{A}}|_{\mathcal{C}_i} \in \Theta(f_i|_{\mathcal{C}_i})$.*

We say that an algorithm is speculative if there exist a problem $\mathcal{P}$, a set of functions $\mathcal{F}$, and a set of classes of dynamic graphs $\mathcal{S}_{\mathcal{C}}$ such that this algorithm is $(\mathcal{P}, \mathcal{F}, \mathcal{S}_{\mathcal{C}})$-speculative. We say that an algorithm is speculative with respect to a family of classes of dynamic graphs $\mathcal{C}_i$, ..., $\mathcal{C}_k$, for a problem $\mathcal{P}$, in function of a category of costs (for instance a memory cost in bytes) if there exists a set of functions $\mathcal{F}$ (each associating to any execution a cost belonging to the previous category of costs) such that this algorithm is $(\mathcal{P}, \mathcal{F}, \{\mathcal{C}_i, \ldots \mathcal{C}_k\})$-speculative.

Above, we give a formal definition of a speculative algorithm for robot networks. This definition is suitable to the intuition given of this notion. Indeed, the points 1 and 2 of the definition permit to indicate that the problem must be studied in a dynamic environment that may change with time, and such that there is a relation between the changes (considering a class of dynamic graph studied, there is at least another class such that the two classes are comparable). Moreover, the combination of points 3 and 4 imposes that a set of comparable functions are considered. A speculative algorithm for a problem should solve the problem in each class of dynamic graphs considered, and more the dynamics decrease more its measurement of performance is low. Besides, the point 4 of the definition imposes that the problem studied is not solvable in a strong class of dynamic graphs (possessing weak connectivity assumptions) with a measurement of performance equal to the one with which it is solvable in a weak class (possessing strong connectivity assumptions). This point is mandatory to avoid the conception of speculative algorithms that consider only classes of dynamic graphs in which the problem studied is solvable with a good measurement of performance and that are speculative only because they cannot succeed to find an appropriate optimized solution to the problem.

8.1.2 State of the Art

The definition of speculative algorithms is general. The speculation can be made on multiple criteria. For instance, one may speculate that executions that are more likely to happen are synchronous, while some others may speculate that executions that are more likely to happen are without fault, ... In this section, we present articles of the state of the art providing speculative algorithms.

Kotla et al. [114] present a speculative BFT (i.e., Byzantine Fault-Tolerant) replicated state machine protocol named Zyzzyva. In a BFT replicated state machine protocol, clients send queries to a server (called primary) that replicates each query on multiple other servers (called replicas). Since the primary and the replicas may be subject to Byzantine faults, the replication is done in order to produce correct results to clients. BFT replicated state machine protocols have to ensure consistency in the answers to clients if at most a third of the servers are Byzantine. For the clients to receive consistent answers, each replica must execute the queries of the clients in the same order. In most of the BFT replicated state machine protocols, the replicas have to execute a 3-phase commit protocol to agree on the ordering of the requests. When executing a 3-phase commit protocol the number of messages sent in the network is quadratic in the number of replicas, and the clients receive their answers in five rounds (in good cases). Here in Zyzzyva, it is assumed speculatively that the replicas will execute the requests in the same order. In good cases, the clients receive their answers in three rounds and the total number of messages sent is linear in the number of replicas. Therefore, here, the speculation is used to reduce the cost in time and in messages sent of usual BFT replicated state machine protocols. However, in case of faulty execution, Zyzzyva restarts the process using a 3-phase commit protocol (to guarantee correct results in each case).

Aublin et al. [9] have also proposed two speculative BFT replicated state machine protocols. They first present a protocol named AZyzzyva which speculates that there is no failure and no asynchronism. They also present a protocol named Aliph in which there are two levels of speculation: they speculate that there is no failure, no asynchronism and no contention; and when a problem is detected, they speculate that there is contention but that there is no failure and no asynchronism.

Dubois and Guerraoui [77] are interested in the speculative aspect of self-stabilizing algorithms (algorithms that tolerate arbitrary faults such that there exists a time t from which these faults no longer occur). When there are faults in the system, a self-stabilizing algorithm is not required to always satisfy the specification of the problem $\mathcal{P}$ it has to solve. Hence, this approach is a non-masking approach: the faults are not masked. After time t there exists a time t' from which the self-stabilizing algorithm satisfies the specification of $\mathcal{P}$. The stabilization time

of a self-stabilizing algorithm corresponds to the maximal duration $t' - t$ over all its execution. The authors introduce the notion of speculative stabilization, where the stabilization must be guaranteed in any execution and optimized in executions that are more likely to happen. As an example, they provide a speculative self-stabilizing mutual exclusion algorithm that stabilizes in an optimal number of rounds for synchronous executions.

Note that there is no speculative algorithm for robot networks evolving in dynamic graphs.

8.2 State of the Art About Exploration in Dynamic Graphs

We want to study speculative algorithms applied to robot networks. On this purpose we study the exploration problem.

In this thesis, we focus on deterministic algorithms. However, there exist some probabilistic exploration algorithms performed in dynamic graphs. For instance, this is the case of the algorithm proposed by Avin et al. [11].

Moreover, in this thesis, we focus on online algorithms (i.e., algorithms assuming that the dynamics of the graphs are not known in advance) which seems to be a more realistic approach than the offline approach (where it is assumed known in advance the dynamics of the graphs: the time of appearance and disappearance of each edge is known in advance). However, there exist some articles dealing with offline deterministic solutions to the exploration problem [106, 1, 79, 126].

In the state of the art, the only dynamic environments in which the exploration problem has been studied are the periodic-edges dynamic graphs and the T-interval connected graphs. We present, in Section 8.2.1, the state of the art about the exploration problem in periodic-edges dynamic graphs, and, in Section 8.2.2, the state of the art about the exploration problem in T-interval connected graphs (refer to Section 2.1.2 for the definitions of these classes of dynamic graphs).

8.2.1 Periodic-edges Dynamic Graphs

In this section, we present the state of the art about the exploration problem in periodic-edges dynamic graphs (graphs in which each of the edges is present periodically).

While studying periodic-edges dynamic graphs, a particular model (that cannot model all the possible periodic-edges dynamic graphs, see Figure 8.1) named PV-graph is considered in the state of the art, hence, we first present briefly this model here. A PV-graph is composed of sites and carriers: the sites are the nodes of the graph, and the carriers are entities that move from site to site in a periodic way. Hence, the edges of a PV-graph appear and disappear depending on the movements of the carriers. Since the movements of the carriers are periodic, the appearance and disappearance of the edges are also periodic. The ordered set of sites visited periodically by a carrier is called a route. Carriers may carry robots (i.e., carriers may transport robots from site to site). This model is inspired by public transportation systems like subways transporting passengers.

Note that the graph formed by all the routes of the carriers is a directed dynamic graph. When the routes formed by the carriers have the same period they are qualified of homogeneous, otherwise they are qualified of heterogeneous. If a route does not contain sites with self-loop or multiple arcs linking two sites, then it is a simple route. If each arc of a route appears at most once in this route, then the route is said to be circular. Finally, if a route is neither simple nor circular, it is said to be arbitrary. In the following, n corresponds to the number of sites in the PV-graph, k is the number of carriers and p is the size of the longest route.

Flocchini et al. [89] consider the PV-graph model where the carriers move synchronously from site to site in a periodic way and in which only a single robot is evolving. Initially the robot is located on a carrier, and it can move from a carrier c_1 to a carrier c_2 only if c_1 and c_2 are

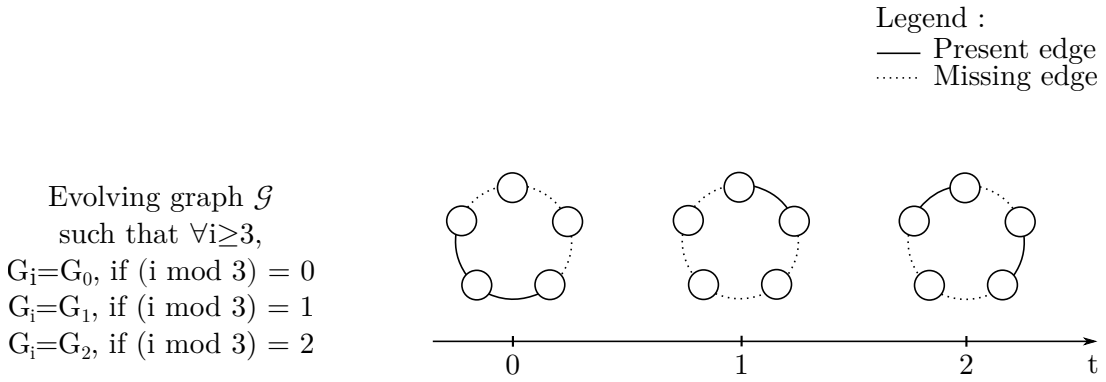


Figure 8.1: Example of a periodic-edges evolving graph that cannot be modeled in the PV-graph model.

Routes	Homogeneous	Heterogeneous
Arbitrary	$\Theta(k * p)$	$\Theta(k * p^2)$
Simple	$\Theta(k * n^2)$	$\Theta(k * n^4)$
Circular	$\Theta(k * n)$	$\Theta(k * n^2)$

Table 8.1: Complexities in terms of movements of the algorithms provided by Flocchini et al. [89] depending on the kind of the routes.

located at the same time on the same site. The carriers have distinct identifiers. The robot is able to know the identifier of the carrier that carries it. In this setting, the authors establish the necessary conditions for the robot to visit each of the sites in finite time and then to terminate its execution (i.e., for the exploration with stop to be solved).

Note that, in this setting, only the exploration of highly connected PV-graphs is possible (where highly connected means that the graph, where nodes represent the carriers and where there exists an edge between two carriers if these two carriers are located at the same time on the same site, is connected).

The authors prove that if the sites have no identifiers, the exploration cannot be solved if the robot has no knowledge on an upper bound on p. They also prove that if the sites have distinct identifiers, then the exploration is impossible if the robot does not know n or an upper bound on p.

The authors present different algorithms to solve the exploration problem in such setting. Depending on the kind of routes (homogeneous, heterogeneous, simple, circular, arbitrary) present in the system, the authors give the different complexities of their algorithms. All their complexities correspond to the number of movements the robot has to do to visit all the sites. See Table 8.1 for the complexities of their algorithms. For each kind of routes, they also compute the lower bound in terms of movements to solve the exploration problem. This study shows that the complexities of their algorithms fit the lower bounds.

Wade and Ilcinkas [105] also consider the PV-graph model where the carriers move synchronously from site to site in a periodic way, and in which only a single robot evolves. However, contrary to Flocchini et al. [89], for the robot to move from a carrier to another one, there is no need for the two carriers to be present at the same time on the same site: the robot can disembark (from the first carrier) on a site and wait for another carrier on this site. This permits the authors to analyze the exploration with stop in a connected PV-graph, where connected means that the graph, where nodes represent the carriers and where there exists an edge between two carriers if there exists a common site on their routes, is connected. They do not need the PV-graph to be

a highly connected PV-graph. They consider that there is no identifier on the sites. The carriers have distinct identifiers and the robot moving on a carrier is able to know its identifier.

In this setting, first, the authors prove (like Flocchini et al. [89] did) that the robot needs some information on the PV-graph, like an upper bound on p, to be able to explore it. Then, the authors provide an algorithm such that, in the general case (heterogeneous and arbitrary routes), its complexity in terms of movements is in $O(min\{k * p, n * p, n^2\})$. The authors, besides, prove that the lower bound in terms of movements to solve the exploration problem fits the complexity obtained by their algorithm. They also show that their algorithm provides a time complexity in $O(n * p)$ rounds when the routes are heterogeneous and arbitrary, and prove that this fits the lower bound of the time complexity needed to solve the exploration problem while considering these kinds of routes. The authors prove that all these results are still true when considering homogeneous or highly connected routes. In other words, their algorithm for homogeneous or highly connected routes has a complexity in terms of movements in $\Theta(min\{k * p, n * p, n^2\})$ and a time complexity in $\Theta(n * p)$ rounds.

Flocchini et al. [88] consider PV-graphs in which multiple robots evolve and where some of the sites (called black holes) make disappear the robots that disembark on them. They assume that there are terminal sites: a terminal site forces the robots carried on this site to disembark on it. The carriers they consider move asynchronously from site to site. They also consider that the robots compute in an asynchronous manner. All the robots of the system start initially their execution on the same site of the PV-graph. The robots must wait on sites to move from one carrier to another one. They communicate with each other using whiteboards on sites that can be read and written by them. Each whiteboard is accessed in fair mutual exclusion. In this setting, the authors study the problem of the exploration such that, in finite time, at least one robot has not disappeared, and all robots that have not disappeared stop their execution and know which sites of the PV-graph are black holes.

First, the authors establish the necessary conditions for this problem to be solvable. Particularly, they prove that the carriers must have distinct identifiers visible by the robots for the exploration problem to be solved. They also prove that if the robots do not know either the number of carriers in the system or the number of sites in the system, then the exploration is impossible. Moreover, for each site s, the robots must know how many routes contained s. The site where all the robots start their execution, as well as the terminal sites must not be black holes. The robots must know the total number of black holes contained in each route. Call this number the number of black hole stops. Since some sites are present in multiple routes, and since some carriers may cross multiple time the same sites while following their routes, note that the number of black hole stops may be greater than the total number of black holes. The number of robots must be strictly greater than the number black hole stops. The graph, where the nodes represent the carriers and where an edge between two carriers means that their routes share a site that is not a black hole, must be connected. The robots must know the length of the routes of each carrier. Finally, the robots must be able, once disembarked on a site s from a carrier c, to go back on c from s.

Then, the authors prove that these necessary conditions are also sufficient. To do so, they provide an algorithm whose complexity in terms of movements is equal to $O(\mathcal{R} * k^2 * p + k * p^2)$ where $\mathcal{R}$ is the number of robots in the system. Finally, they prove that the complexity of their algorithm fits the lower bound.

Flocchini et al. [87] are interested in the exact same problem as Flocchini et al. [88], except that they assume that the robots may start from arbitrary scattered positions. With the exact same assumptions than those considered by Flocchini et al. [88], except that the whiteboards are now on the carriers and not on the sites, they provide an algorithm whose complexity in terms of movements is equal to $O(\mathcal{R} * k^2 * p + k * p^2)$. Therefore, this algorithm is optimal (since its complexity matches the lower bound provided by Flocchini et al. [88] and since the proof to show this lower bound is independent of where the whiteboards are located).

Value of T	Complexity
$T = 1$	$\Theta(n)$
$2 \leq T < (n+1)/2$	$\Theta(n - T)$
$T \geq (n+1)/2$	$\Theta(n)$

Table 8.2: Time complexities (in rounds) of the algorithm provided by Ilcinkas and Wade [106] depending on the value T defining the T-interval connected ring in which a robot (knowing the dynamics of the ring) evolves.

8.2.2 T-interval connected Graphs

In this section, we present the state of the art about the exploration problem in T-interval connected graphs (graphs connected at each instant time such that, for all-time interval of length T, a subset of the edges present form the same spanning connected subgraph of G, where G is the footprint of the dynamic graph). While considering T-interval connected graphs, some articles of the state of the art make the strong assumption that the exploring robots know the dynamics of the graphs in which they evolve, i.e., the robots know which edges appear and disappear and at which time. Some articles, however, assume that the robots do not have access to such knowledge. We first present below the articles that deal with robots knowing the dynamics of the graphs and then we present the articles where the robots have no knowledge on the dynamics.

With knowledge of the dynamics. Ilcinkas and Wade [106] consider only a single robot evolving in T-interval connected rings. In this setting, they provide an algorithm exploring any ring in $O(n)$ rounds. More precisely, depending on the value T defining the T-interval connected ring in which the robot is evolving, the time complexity of their algorithm varies (refer to Table 8.2). The authors prove that the time complexities obtained by their algorithm fit the lower bounds.

Ilcinkas et al. [104] consider only a single robot evolving in T-interval connected cactus (i.e., connected graphs in which any two simple cycles have at most one node in common). In this setting, they provide an algorithm exploring any cactus in $O(2^{\sqrt{log(n)}} * n)$ rounds. The authors prove that the time complexity obtained by their algorithm fits the lower bound.

Without knowledge of the dynamics. Ilcinkas and Wade [106] consider only a single robot evolving in $\mathcal{BRE}$ and T-interval connected rings, i.e., even if at most one edge of the ring may be missing at each instant time, any edge appears at least once every δ units of time. In this setting, the authors search for the necessary and sufficient time to explore a dynamic ring. They provide an algorithm solving the exploration problem in $O(n+(n/max\{1,T\})*\delta)$ rounds, and prove that this time complexity fits the lower bound.

Flocchini et al. [86] consider arbitrary undirected graphs in which some nodes (called black holes) make disappear the robots that are located on them, and some edges (called black edges) make disappear the robots that cross them. The black holes and edges are fixed in the network: their positions do not evolve with time. In other words, the dynamics do not come from the black holes and edges. The nodes that are not black holes are called white nodes, and the edges that are not black are called white edges. Among the white edges, some may fail during the execution. The dynamics come from these failures. An edge cannot fail while a robot is crossing it. The edges may fail at unpredictable time, but once an edge has failed, it does not appear anymore in the graph (i.e., it cannot recover). Any number of edges may fail as long as the white nodes and white non-faulty edges form a connected graph. Hence, the dynamics modeled correspond to particular $\mathcal{AC}$ graphs (i.e., 1-interval connected graphs) where no edge that disappears can appear again.

While focusing on this environment, the authors consider asynchronous robots that are able to communicate thanks to whiteboards on nodes in which robots can read and write in fair mutual exclusion. All the robots have distinct identifiers. The authors assume that the edges are crossed in a FIFO way, and that nodes and ports of a same node have distinct identifiers. They study the exploration of the graph starting from a configuration where all the robots are scattered. They indicate that the exploration is successful if, in finite time, the four following conditions are true: (*i*) at least one robot has not disappeared; (*ii*) all the robots that have not disappeared terminate their execution; (*iii*) all the (non-faulty) white edges linking two white nodes have been crossed; and (*iv*) all the black edges linking a white node to any other node and white edges linking a white node to a black hole have been marked as dangerous (i.e., the robots have indicated on the whiteboards that these edges must not be crossed). The authors show that solving the exploration is possible. Moreover, thanks to their algorithm, they show that the exploration can be done thanks to $f + 1$ robots, where f is equal to the number of edges (black or white) linking a black hole and a white node, plus two times the number of black edges linking two white nodes.

Di Luna et al. [125] consider undirected $\mathcal{AC}$ rings (which are 1-interval connected rings). The nodes of the rings are divided into three parts: one for each port of the node and one at the middle of the node. The robots are synchronous (either fully-synchronous or semi-sychronous) and have access to each port of each node in mutual exclusion (i.e., at each instant time there is at most one robot on each port of each node). The robots are anonymous. When a robot is on a node, it is not able to detect if the adjacent edges to its node are present or missing. The authors consider multiple assumptions on the robots and on the ring and determine, depending on these assumptions, the feasibility of the exploration with stop problem (in which the robots have to explore in finite time the graph, and then they are required, also in finite time, to stop their execution once they detect that the graph has been explored). First they consider the fully-synchronous model. In this model, they prove that two robots are not able to solve the exploration with stop problem if the ring is anonymous (the nodes cannot be distinguished) and if the robots do not know an upper bound on the size of the ring. They prove, when the ring is anonymous and the robots know an upper bound N on the size of the ring, that the exploration with stop problem is solvable in $O(N)$ rounds. This result is true whether the robots have the same chirality or not. When the robots do not know an upper bound on the size of the ring, the exploration is possible if there is a marked node (i.e., a node that is distinguishable from the other nodes). Indeed, in this setting, the authors provide an algorithm that solves the exploration with stop problem in $O(n)$ (resp. $O(n * log(n))$) rounds, when the robots have (resp. have not) the same chirality. All the algorithms provided use only two robots.

The authors also analyze the feasibility of this problem when the robots are semi-synchronous. They study three different settings:

Passive transportation model: in this model, a sleeping robot (i.e., a robot that it is not selected to execute its L-C-M cycle) at a round t, located on the port p of its current node, will be moved during the Move phase of round t on the adjacent node linked to p in case where the adjacent edge to p is present at time t.

Eventual transportation model: in this model, a sleeping robot at a round t, located on the port p of its current node, will eventually be selected to execute its L-C-M cycle while the adjacent edge to p is present.

No simultaneity model: a sleeping robot located on the port p of its current node is not moved passively, and there is no guarantee on a possible selection of this robot to execute its L-C-M cycle while the adjacent edge to p is present.

While considering the no simultaneity model, the authors prove that whatever the number of robots present in the system, the exploration with stop problem is not solvable.

While considering the eventual transportation model, they also prove that the exploration with stop problem is not solvable whatever the number of robots. This is true even if the robots

know an upper bound on the size of the ring, have the same chirality, possess distinct identifiers and the ring possesses one marked node. However, when the exact size of the ring is known by the robots, the authors provide an algorithm solving the exploration with stop problem using three robots without chirality.

While considering the passive transportation model, they prove that two robots without chirality knowing the size of the ring cannot solve the exploration with stop problem. However, in this setting, they provide an algorithm using two robots knowing an upper bound N on the size of the ring and having the same chirality that succeeds to solve the exploration with stop problem in $O(N^2)$ moves. They prove that the lower bound to solve this problem is in $\Omega(n * N)$ moves. Finally, they provide an algorithm solving the exploration with stop problem using three robots without chirality and knowing an upper bound N on the size of the ring in $O(N^2)$ moves.

Gotoh et al. [99] consider robots evolving in undirected dynamic tori of size $n * m$ with $3 \leq n \leq m$, a torus being a set of n row rings and m column rings where each node is implied in two different rings (intuitively, it is a grid where each extremity node is linked to the opposite extremity node of the same row and/or column if any). More precisely, the dynamic tori they study are particular $\mathcal{AC}$ tori since each of the rings composing a torus are $\mathcal{AC}$ rings. The authors assume that each of the nodes of a torus is labeled with its row and column number. Each of the ports of a node is labeled with $right$, $left$, $down$ and up. The labels of the nodes and ports are visible by the robots. Each node is divided into five parts: one for each of the ports of the node and one at the middle of the node. The robots have the same chirality. They are fully-synchronous. The robots access to each port of each node in mutual exclusion (i.e., at each instant time there is at most one robot on each port of each node). They know the size of the torus in which they are evolving: they know n and m. A robot is able to see if there are other robots on the different parts of its current node, but it is not able to know the exact number of robots located on each part of its node. The authors study the necessary and sufficient number of robots to explore with stop any torus when the robots are able to see the presence/absence of the adjacent edges of their current node and when they are not endowed with this ability.

When the robots are not able to detect the presence/absence of the adjacent edges of their current node, the authors prove that n or less robots are not able to explore any dynamic torus (of size $n * m$ with $3 \leq n \leq m$). They provide two general algorithms for this setting: one solving the exploration of any dynamic torus with $n+2$ robots (or more) in $O(m^2)$ rounds, and one using $n+1$ robots (or more) solving the exploration of any dynamic torus in $O(n * m^2)$ rounds. Finally, they provide an algorithm for this setting in the particular case where $n = m$, this algorithm solves the exploration with $n + 1$ robots in $O(n^2)$ rounds.

When the robots are able to detect the presence/absence of the adjacent edges of their current node, the authors prove that when $n = 4$, at least 4 robots are needed to explore the dynamic tori, and when $n > 4$, then at least $\lceil n/2 \rceil + 1$ robots are needed to explore the dynamic tori. They provide two general algorithms for this setting: one solving the exploration of any dynamic torus with $n \geq 4$ using $\lceil n/2 \rceil + 2$ robots (or more) in $O(m^2)$ rounds, and one using $\lceil n/2 \rceil + 1$ robots (or more) solving the exploration of any dynamic torus with $n \geq 5$ in $O(n * m^2)$ rounds. Finally, they provide an algorithm for this setting in the particular case where $n = m$ and $n \geq 5$, this algorithm solves the exploration using $\lceil n/2 \rceil + 1$ robots in $O(n^2)$ rounds.

8.3 Contributions

There is no speculative algorithm using robots evolving in dynamic graphs. The goal of this part is to extend the speculative notion to robot networks evolving in dynamic graphs. On this purpose, we choose to study the exploration problem. As seen in Section 8.2, this problem has never been analyzed in $\mathcal{COT}$ graphs. It has also never been studied in the case where robots are subject to transient faults (i.e., arbitrary faults such that there exists a time from which these faults no longer occur). To fill the lack of the state of the art, in this part, our contribution is

about speculative algorithms for the perpetual exploration problem using non-faulty as well as self-stabilizing (i.e., that tolerate transient faults) robots.

The perpetual exploration problem for dynamic graphs may be formalized as follows:

Definition 8.2 (Perpetual exploration of an evolving graph)**.** *Given an evolving graph $\mathcal{G}$, a perpetual exploration algorithm guarantees that every node of $\mathcal{G}$ is infinitely often visited by at least one robot (i.e., a robot is infinitely often located at every node of $\mathcal{G}$).*

Note that this specification of the perpetual exploration does not require that every robot visits infinitely often every node of $\mathcal{G}$.

In Chapter 9, we first characterize the necessary number of robots not subject to faults permitting to solve the perpetual exploration problem in $\mathcal{COT}$ rings. Then, we prove, by construction, that it is possible to solve the perpetual exploration problem in $\mathcal{COT}$ rings with robots that are not subject to faults. Since our algorithm solves the perpetual exploration problem under the strongest class of dynamic rings (of the classification of Casteigts et al. [40, 38]) having a recurrent temporal connectivity, it is interesting to study the existence of a speculative algorithm solving this problem. The algorithm we provide is besides a speculative algorithm. To prove the speculative aspect of our algorithm we have to prove that, under some weak classes of dynamic rings, it provides some good execution properties. Here, we focus on the optimization of the time complexity in $\mathcal{ST}$ rings (compared to the one in $\mathcal{COT}$ rings). Since we study the perpetual exploration problem, it is not possible to optimize the time complexity corresponding to the amount of time taken for the task to terminate. We, therefore, focus on the cover time which is defined formally below.

Definition 8.3 (Cover time)**.** *The cover time of an execution e is the worst time of the minimal time taken by the robots to explore at least once each node of the graph from any time of e. The cover time of an algorithm $\mathcal{A}$ is the worst cover time of the executions of $\mathcal{A}$ allowed by the model.*

While considering a $\mathcal{COT}$ ring, it is possible for all the edges of the graph to be initially missing and to stay missing for any arbitrary long time. Hence, since generally there are more nodes in the rings than robots to explore them, the cover time of every algorithm solving the perpetual exploration problem cannot be bounded when robots evolve in $\mathcal{COT}$ rings. Therefore, to prove the speculative aspect of our algorithm, we have to prove that its cover time in $\mathcal{ST}$ rings is bounded. We will additionally show that the cover time of our algorithm is asymptotically optimal in $\mathcal{ST}$ rings.

In Chapter 10, we adopt the same strategy as in Chapter 9 except that we consider robots that may be subject to faults. More precisely, we consider robots subject to transient faults. Algorithms that are able to tolerate these kinds of faults are called self-stabilizing algorithms.

Finally, in Chapter 11, we conclude the part about the speculation on the perpetual exploration. Then, we give an extension of our work on the characterization of the necessary and sufficient number of robots permitting to solve the perpetual exploration problem considering graphs other than ring-shaped ones. However, for these kinds of graphs we have not considered the speculative aspect of the algorithm given.

Chapter 9

Perpetual Exploration Without Fault

Contents

The exploration problem is an important problem of the literature. It has been largely studied in static graphs and has been considered in some dynamic graphs ($\mathcal{AC}$, T-interval connected, *etc.*) but has never been studied in $\mathcal{COT}$ graphs. In this chapter, we characterize the number of robots necessary to solve the perpetual exploration problem in $\mathcal{COT}$ rings. Depending on these results, we describe some algorithms showing that the necessary number of robots is also sufficient to solve the perpetual exploration problem in $\mathcal{COT}$ rings. Even if it is possible to solve the perpetual exploration problem in $\mathcal{COT}$ rings, the cover time of such algorithms is necessarily unbounded (refer to Section 8.3). This motivates the conception of a speculative algorithm to solve the perpetual exploration problem. Indeed a speculative algorithm is an algorithm that must satisfy its requirements (i.e., the specification of the problem considered) in any execution in which it is susceptible to be executed but also that must be very efficient for the executions that are more likely to happen. In this chapter we consider only $\mathcal{ST}$ graphs (in addition to $\mathcal{COT}$ graphs) to prove the speculative aspect of our algorithm, i.e., we prove that the cover time of our algorithm is bounded (it has a cover time in $\Theta(n-R)$ rounds) and is asymptotically optimal when executed in $\mathcal{ST}$ graphs.

The results presented in this chapter have been published in ICDCS 2017 [29], and in ALGOTEL 2017 [30].

Results. In this chapter, we analyze the computability of the perpetual exploration problem in $\mathcal{COT}$ rings, i.e., we study the deterministic solvability of this problem with respect to the number of robots. In other words, we establish the necessary and sufficient number of robots to solve the perpetual exploration problem for any size of $\mathcal{COT}$ rings (see Table 9.1 for a summary). Our results for the sufficiency are constructive: we give algorithms and their proofs of correctness to prove the sufficiency. The algorithm using 3 robots and more is a speculative algorithm: its cover time is unbounded when executed in $\mathcal{COT}$ rings, however, when executed in $\mathcal{ST}$ rings, its cover time is bounded (in $\Theta(n-R)$ rounds) and asymptotically optimal.

In more details, we first provide, in Section 9.1, an algorithm that perpetually explores, using a team of $\mathcal{R} \geq 3$ robots, any $\mathcal{COT}$ ring of $n > \mathcal{R}$ nodes. We prove, in Section 9.2, that this algorithm is speculative, and that its cover time is asymptotically optimal when executed in

Number of Robots	Size of Rings	Results
3 and more	≥ 4	Possible (Theorem 9.1)
2	> 3	Impossible (Theorem 9.3)
	$= 3$	Possible (Theorem 9.4)
1	> 2	Impossible (Theorem 9.5)

Table 9.1: Overview of the results.

$\mathcal{ST}$ rings. Then, we show, in Section 9.3.1, that two robots are not sufficient to perpetually explore $\mathcal{COT}$ rings with a number of nodes strictly greater than three. We provide, in Section 9.3.2, an algorithm using two robots that perpetually explore 3-nodes $\mathcal{COT}$ rings. In Section 9.4, we show that a single robot cannot perpetually explore $\mathcal{COT}$ rings with a number of nodes strictly greater than two. Finally, Section 9.5 concludes the chapter.

In this chapter, we consider a system made of $\mathcal{R}$ anonymous robots evolving in $\mathcal{COT}$ rings. Since the robots are anonymous, if multiple robots start from the same node they act similarly (if they have the same chirality). This is a problem since a single robot is not able to solve the exploration problem in $\mathcal{COT}$ rings (refer to Theorem 9.5). Therefore, we assume that initially the robots start their execution at distinct positions in the dynamic graph. Note that, since we assume that initially the robots are on distinct positions, we suppose that $\mathcal{R} < n$, in order not to have the ring initially trivially explored. In other words, we define a well-initiated execution as an execution $(\gamma_0, \gamma_1)(\gamma_1, \gamma_2)(\gamma_2, \gamma_3)\ldots$ such that γ_0 contains strictly less robots than the number of nodes of $\mathcal{G}$ and is towerless (i.e., there is no tower in this configuration). In addition to the common assumptions made on robots all along the chapters of this thesis (see Section 3.2), we assume that the robots do not have the same chirality. The robots have no prior knowledge about the graph in which they evolve nor on the robots. They are unable to directly communicate with each other by any means. They are endowed with weak local multiplicity detection meaning that they are able to detect if they are alone on their current node or not, but they cannot know the exact number of co-located robots. Call this model $\mathcal{M}_{exploration}$ (also refer to Figure 3.5).

For the algorithms presented in this chapter, the variable *dir* of the robots can only take the values $right$ or $left$ (i.e., the variable *dir* cannot be equal to $\perp$). Initially, this variable is set to $left$ (which does not necessarily correspond to the counter-clockwise direction since the robots do not necessarily have the same chirality).

9.1 With Three or More Robots

This section is dedicated to the more general result: the perpetual exploration of $\mathcal{COT}$ rings of size greater than k (i.e., $n > k$) with a team of $k \geq 3$ robots. The speculative aspect of this algorithm is studied in Section 9.2.

9.1.1 Presentation of the Algorithm

We first describe intuitively the key ideas of our algorithm. Remind that an algorithm controls the move of the robots through their variable direction. Hence, designing an algorithm consists in choosing when we want a robot to keep its direction and when we want it to change its direction (in other words, turn back).

The first idea of our algorithm is to require that a robot keeps its direction when it is not involved in a tower (Rule $\mathbf{R_1}$). Using this idea, some towers are necessarily formed when there exists an eventual missing edge. Our algorithm reacts as follows to the formation of towers. If at a time t a robot does not move and forms a tower at time $t + 1$, then the algorithm keeps the

direction of the robot (Rule $\mathbf{R_2}$). In the contrary case (that is, at time t, the robot moves and forms a tower at time t+1) it changes the direction of the robot (Rule $\mathbf{R_3}$).

Let us now explain how the algorithm (Rules $\mathbf{R_1}$, $\mathbf{R_2}$, and $\mathbf{R_3}$) enables the perpetual exploration of any $\mathcal{COT}$ ring. First, note that Rule $\mathbf{R_1}$ alone is sufficient to perpetually explore $\mathcal{COT}$ rings without eventual missing edge provided that the robots never meet (since in this case a robot does not change its direction and is infinitely often able to move in that direction). The main property induced by Rules $\mathbf{R_2}$ and $\mathbf{R_3}$ is that any tower is broken in a finite time and that at least one robot of the tower considers each possible direction. This property implies (combined with Rule $\mathbf{R_1}$) that (i) the algorithm is able to perpetually explore any $\mathcal{COT}$ ring without eventual missing edge (even if robots meet); and that (ii), when the ring contains an eventual missing edge, one robot is eventually located at each extremity of the eventual missing edge and considers afterwards the direction of the eventual missing edge.

Let us consider this last case. We call sentinels the two robots located at extremities of the eventual missing edge. The other robots are called explorers. By Rule $\mathbf{R_3}$, an explorer that arrives on a node where a sentinel is located changes its direction. Intuitively, that means that the sentinel signal to the explorer that it has reached one extremity of the eventual missing edge and that it has consequently to turn back to continue the exploration. Note that, by Rule $\mathbf{R_2}$, the sentinel keeps its direction (and hence its role). Once an explorer leaves an extremity of the eventual missing edge, we know, thanks to Rule $\mathbf{R_1}$ and the main property induced by Rules $\mathbf{R_2}$ and $\mathbf{R_3}$, that a robot reaches in a finite time the other extremity of the eventual missing edge and that (after the second sentinel/explorer meeting) all the nodes have been visited by a robot in the meantime. As we can repeat this scheme infinitely often, our algorithm is able to perpetually explore any $\mathcal{COT}$ ring with an eventual missing edge, that ends the informal presentation of our algorithm.

Refer to Algorithm 9.2 for the formal statement of our algorithm called $\mathbb{PEF}\_3+$ (standing for $\mathbb{P}$erpetual $\mathbb{E}$xploration in $\mathbb{F}$SYNC with 3 or more robots). Note that we have explained intuitively the algorithm thanks to 3 rules, but the algorithm is written with only one rule. Rule **Robots** of $\mathbb{PEF}\_3+$ corresponds to Rule $\mathbf{R_3}$ of the intuitive description. The two other rules (Rules $\mathbf{R_1}$ and $\mathbf{R_2}$ of the intuitive description) do not change the value of the variable *dir* of the robots, and hence there is no need for them to be written: the absence of the execution of Rule **Robots** corresponds to the execution of either Rule $\mathbf{R_1}$ or Rule $\mathbf{R_2}$.

Algorithm 9.1 $\mathbb{PEF}\_3+$

1: **Robots** :: $HasMoved() \wedge ExistsOtherRobotsOnNode() \longrightarrow dir := \overline{dir}$

9.1.2 Proof of Correctness

In this section, we prove the correctness of $\mathbb{PEF}\_3+$ with $k \geq 3$ robots. In the following, we consider a $\mathcal{COT}$ ring $\mathcal{G}$ of size at least $k+1$. Let $\varepsilon = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)(\gamma_2, \gamma_3)\ldots$ be any execution of $\mathbb{PEF}\_3+$ in $\mathcal{G}$.

To prove the correctness of our algorithm in $\mathcal{COT}$ rings, we consider multiple cases: we first study $\mathcal{COT}$ rings without eventual missing edge (refer to Lemmas 9.2 and 9.5), then we consider $\mathcal{COT}$ rings with one eventual missing edge (refer to Lemma 9.8). When studying $\mathcal{COT}$ rings without eventual missing edge, we analyze the case where there is no tower in the execution (Lemma 9.2) and the case where there are towers in the execution (Lemma 9.5). We use some other lemmas to prove these three main lemmas: in particular we use lemmas giving properties on towers, and lemmas giving properties on configurations reached when an eventual missing edge is present in the execution.

Lemma 9.1. *If there exists an eventual missing edge in $\mathcal{G}$, then at least one tower is formed in ε.*

Proof. By contradiction, assume that e is an eventual missing edge of $\mathcal{G}$ (such that e is not present in $\mathcal{G}$ after time t) and that no tower is formed in ε.

Executing $\mathbb{PEF}\_3+$, a robot changes the global direction it considers only when it forms a tower with another robot. As, by assumption, no tower is formed in ε, each robot is always considering the same global direction. All the edges of $\mathcal{G}$, except e, are infinitely often present in $\mathcal{G}$. Hence, any robot reaches one of the extremities of e in finite time after t. As the robots consider a direction at each instant time and that there are at least 3 robots, at least 2 robots consider the same global direction at each instant time. Hence, at least two robots reach the same extremity of e. A tower is formed, leading to a contradiction. □

Lemma 9.2. *If ε does not contain a tower, then every node is infinitely often visited by a robot in ε.*

Proof. Assume that there is no tower formed in ε. By Lemma 9.1, if there is an eventual missing edge in $\mathcal{G}$, then there is at least one tower formed. In consequence, all the edges of $\mathcal{G}$ are infinitely often present in $\mathcal{G}$.

Executing $\mathbb{PEF}\_3+$, a robot changes the global direction it considers only when it forms a tower with another robot. Hence, none of the robots change the global direction it considers in ε. Since all the edges are infinitely often present, each robot moves infinitely often in the same global direction, that implies the result. □

Lemma 9.3. *If a tower T of 2 robots is formed in ε, then these two robots consider two opposite global directions while T exists.*

Proof. Assume that 2 robots form a tower at a time t in ε. Let us consider the 2 following cases:

Case 1: The two robots consider the same global direction during the Move phase of time t − 1.

In this case, one robot (denoted r) does not move during the Move phase of time t, while the other (denoted r') moves and joins the first one on its current node. During the Compute phase of time t, r still considers the same global direction, while r' changes the global direction it considers by construction of $\mathbb{PEF}\_3+$. Then, the two robots consider two different global directions after the Compute phase of time t.

Case 2: The two robots consider two opposite global directions during the Move phase of time t − 1.

In this case, the two robots move at time $t-1$. During the Compute phase of time t, the two robots change the global direction they consider by construction of $\mathbb{PEF}\_3+$. Hence they consider two different global directions after the Compute phase of time t.

A robot executing $\mathbb{PEF}\_3+$ changes its global direction only if it has moved during the previous step. So, the robots of the tower do not change the global direction they consider as long as they are involved in the tower. As the two robots consider two different global directions after the Compute phase of time t, we obtain the lemma. □

Lemma 9.4. *No tower of ε involves 3 robots or more.*

Proof. We prove this lemma by recurrence. As there is no tower in γ_0 by assumption, it remains to prove that, if γ_t contains no tower with 3 or more robots, so is γ_{t+1}. Let us study the following cases:

Case 1: γ_t contains no tower.

The robots can cross at most one edge at each step. Each node has at most 2 adjacent edges in G_t, hence the maximum number of robots involved in a tower of γ_{t+1} is 3. If a tower

involving 3 robots is formed in γ_{t+1}, one robot r has not moved during the Move phase of time t, while the two other robots (located on the two adjacent nodes of its location) have moved to its position. That implies that the two adjacent edges of the node where r is located are present in G_t. As any robot considers a global direction at each instant time, r necessarily moves in step t, that is contradictory. Therefore, only towers of 2 robots can be formed in γ_{t+1}.

Case 2: γ_t contains towers of at most 2 robots.

Let T be a tower involving 2 robots in γ_t and u be the node where T is located in γ_t. By Lemma 9.3, the 2 robots of T consider two opposite global directions in γ_t.

Consider the 3 following sub-cases:

(i) If there is no adjacent edge to u in G_t, then no other robot can increase the number of the robots involved in the tower.

(ii) If there is only one adjacent edge to u in G_t, then only one robot may traverse this edge to increase the number of robots involved in T. Indeed, if there are multiple robots on an adjacent node to u, then these robots are involved in a tower T' of 2 robots (by assumption on γ_t) and they are considering two opposite global directions in γ_t. However, as an adjacent edge to u is present in G_t and as the robots of T are considering two opposite global directions, then one robot of T leaves T at time t. In other words, even if a robot of T' moves on u, one robot of T leaves u. Then, there is at most 2 robots on u in γ_{t+1}.

(iii) If there are two adjacent edges to u in γ_t, then, using similar arguments as above, we can prove that only one robot crosses each of the adjacent edges of u. Moreover, the robots of T move in opposite global directions and leave u, implying that at most 2 robots are present on u in γ_{t+1}.

□

Lemma 9.5. *If $\mathcal{G}$ has no eventual missing edge and ε contains towers then every node is infinitely often visited by a robot in ε.*

Proof. Assume that $\mathcal{G}$ has no eventual missing edge and ε contains towers.

We want to prove the following property. If during the Look phase of time t, a robot r is located on a node u considering the global direction gd, then there exists a time $t' \geq t$ such that, during the Look phase of time t', a robot is located on the node v adjacent to u in the global direction gd and considers the global direction gd.

Let $t" \geq t$ be the smallest time after time t where the adjacent edge of u in the global direction gd is present in $\mathcal{G}$. As all the edges of $\mathcal{G}$ are infinitely often present, $t"$ exists.

(i) If r crosses the adjacent edge of u in the global direction gd during the Move phase of time $t"$, then the property is verified.

(ii) If r does not cross the adjacent edge of u in the global direction gd, this implies that r changes the global direction it considers during the Look phase of time t. While executing $\mathbb{PEF}\_3+$, a robot changes its global direction when it forms a tower with another robot. Therefore, at time t, r forms a tower with a robot r'. By Lemmas 9.4 and 9.3, two robots involved in a tower consider two opposite global directions. Hence, after the Compute phase of time t, r' considers the global direction gd. A robot executing $\mathbb{PEF}\_3+$ does not change the global direction it considers until it moves. So, r' considers the global direction gd during the Move phase of time $t"$. Hence, during the Look phase of time $t" + 1$, r' is on node v and considers the global direction gd.

By applying recurrently this property to any robot, we prove that all the nodes are infinitely often visited. □

Lemma 9.6. *If $\mathcal{G}$ has an eventual missing edge e (such that e is missing forever after time t) and, during the Look phase of a time $t' \geq t$, a robot considers a global direction gd and is located on a node at a distance $d \neq 0$ in $U_{\mathcal{G}}^{\omega}$ from the extremity of e in the global direction gd, then it exists a time $t" \geq t'$ such that, during the Look phase of time $t"$, a robot is on a node at distance $d-1$ in $U_{\mathcal{G}}^{\omega}$ from the extremity of e in the global direction gd and considers the global direction gd.*

Proof. Assume that $\mathcal{G}$ has an eventual missing edge e (such that e is missing forever after time t) and that, during the Look phase of time $t' \geq t$, a robot r considers a global direction gd and is located on a node u at distance $d \neq 0$ in $U_{\mathcal{G}}^{\omega}$ from the extremity of e in the global direction gd.

Let v be the adjacent node of u in the global direction gd.

Let $t" \geq t'$ be the smallest time after time t' where the adjacent edge of u in the global direction gd is present in $\mathcal{G}$. As all the edges of $\mathcal{G}$ except e are infinitely often present and as u is at a distance $d \neq 0$ in $U_{\mathcal{G}}^{\omega}$ from the extremity of e in the global direction gd, then the adjacent edge of u in the global direction gd is infinitely often present in $\mathcal{G}$. Hence, $t"$ exists.

(i) If r crosses the adjacent edge of u in the global direction gd during the Move phase of time $t"$, then the property is verified.

(ii) If r does not cross the adjacent edge of u in the global direction gd, this implies that r changes the global direction it considers during the Look phase of time t. While executing $\mathbb{PEF}\_3+$ a robot changes the global direction it considers when it forms a tower with another robot. Therefore, at time t, r forms a tower with a robot r'. By Lemmas 9.4 and 9.3, two robots involved in a tower consider two opposite global directions. Hence, after the Compute phase of time t, r' considers the global direction gd. A robot executing $\mathbb{PEF}\_3+$ does not change the global direction it considers until it moves. Therefore, r' considers the global direction gd during the Move phase of time $t"$. Hence, during the Look phase of time $t" + 1$, r' is on node v and considers the global direction gd. □

Lemma 9.7. *If $\mathcal{G}$ has an eventual missing edge e, then eventually one robot is forever located on each extremity of e pointing to e.*

Proof. Assume that $\mathcal{G}$ has an eventual missing edge e such that e is missing forever after time t.

First, we want to prove that a robot reaches one of the extremities of e in a finite time after t and points to e at this time. If it is not the case at time t, then there exists at this time a robot considering a global direction gd and located on a node u at distance $d \neq 0$ in $U_{\mathcal{G}}^{\omega}$ from the extremity of e in the global direction gd. By applying d times Lemma 9.6, we prove that, during the Look phase of a time $t' \geq t$, a robot (denote it r) reaches the extremity of e in the global direction gd from u (denote it v and let v' be the other extremity of e), and that this robot considers the global direction gd. Let us consider the following cases:

Case 1: r is isolated on v at time t′.

In this case, by construction of $\mathbb{PEF}\_3+$, r does not change, during the Compute phase of time t', the global direction that it considers during the Move phase of time $t'-1$. Moreover, a robot can change the global direction it considers only if it moves during the previous step. All the edges of $\mathcal{G}$ except e are infinitely often present. As, at time t', r points to e, it cannot move. Therefore, from time t', r does not move and does not change the global direction it considers. Then, r remains located on v forever after t' considering gd.

Case 2: r is not isolated on v at time t′.

By Lemmas 9.4, r forms a tower with only one another robot r'. By Lemmas 9.4 and 9.3, two robots that form a tower consider two opposite global directions. Hence, either r or r' considers the global direction gd while the other one considers the global direction $\overline{gd}$. As all the edges of $\mathcal{G}$ except e are infinitely often present, then in finite time either r or r' leaves

v. We can now apply the same arguments as those used in Case 1 to the robot that stays on v to prove that this robot remains located on v forever after t' considering gd.

In both cases, a robot remains forever on v considering gd after t'. Assume without loss of generality that it is r. Let us consider the two following cases:

Case A: It exists $\mathbf{r' \neq r}$ considering $\overline{\mathbf{gd}}$ at time $\mathbf{t'}$.

We can apply recurrently Lemma 9.6, and the arguments above to prove that a robot is eventually forever located on v' considering $\overline{gd}$.

Case B: All robots $\mathbf{r' \neq r}$ consider gd at time t.

We can apply recurrently Lemma 9.6 to prove that, in finite time, a robot forms a tower with r on v. Then, by construction of $\mathbb{PEF}\_3+$, this robot considers $\overline{gd}$ after the Compute phase of this time (and hence during the Look phase of the next time). We then come back to Case A.

In both cases, the lemma holds. □

Lemma 9.8. *If $\mathcal{G}$ has an eventual missing edge and ε contains towers, then every node is infinitely often visited.*

Proof. Assume that $\mathcal{G}$ has an eventual missing edge e that is missing forever after time t. By Lemma 9.7, there exists a time $t' \geq t$ after which two robots r_1 and r_2 are respectively located on the two extremities of e and pointing to e. As there are at least 3 robots, let r be a robot (located on a node u considering a global direction gd) such that $r \neq r_1$ and $r \neq r_2$. Let v be the extremity of e in the direction gd of u and v' be the other extremity of e.

Applying recurrently Lemma 9.6, we prove that, in finite time, all the nodes between node u and v in the global direction gd are visited and that a robot reaches v. When this robot reaches v, it changes its direction (hence considers $\overline{gd}$) by construction of $\mathbb{PEF}\_3+$ since it moves during the previous step and forms a tower.

We can then repeat this reasoning (with v and v' alternatively in the role of u and with v' and v alternatively in the role of v) and prove that all nodes are infinitely often visited. □

Lemmas 9.2, 9.5, and 9.8 directly imply the following result:

Theorem 9.1. $\mathbb{PEF}\_3+$ *satisfies the perpetual exploration problem under $\mathcal{M}_{exploration}$ in $\mathcal{COT}$ rings of arbitrary size strictly greater than the number of robots using an arbitrary number (greater than or equal to 3) of robots.*

9.2 Speculative Aspect of $\mathbb{PEF}\_3+$

In this section we prove that $\mathbb{PEF}\_3+$ is a speculative algorithm. As indicated in Section 8.3, the cover time of any perpetual exploration algorithm in $\mathcal{COT}$ rings is unbounded (since it is possible, in a $\mathcal{COT}$ graph, to have initially and during an arbitrary long time all the edges of the graph missing). Since, by Theorem 9.1, $\mathbb{PEF}\_3+$ solves the perpetual exploration problem in $\mathcal{COT}$ rings, and since its cover time in $\mathcal{COT}$ rings is unbounded, the following lemma is sufficient to prove that $\mathbb{PEF}\_3+$ is a speculative algorithm for the perpetual exploration problem with respect to $\mathcal{ST}$ and $\mathcal{COT}$, in function of the cover time in rounds. Indeed, in the lemma below, we prove that the cover time of $\mathbb{PEF}\_3+$ in $\mathcal{ST}$ rings is bounded.

We recall that, in this chapter, we assume that $\mathcal{R} < n$ for all the algorithms considered in order to ensure that the perpetual exploration problem is not trivially solved in the initial configuration (due to the initial scattering assumption).

Lemma 9.9. *Our algorithm* $\mathbb{PEF}\_3+$ *under* $\mathcal{M}_{exploration}$*, using* $\mathcal{R}$ *robots in any* $\mathcal{ST}$ *ring of size* n *has a cover time in* $O(n - \mathcal{R})$ *rounds.*

Proof. Assume that initially x robots consider the clockwise direction while y robots consider the counter-clockwise direction. Note that initially all the robots start their execution on distinct nodes. Moreover, by Lemma 9.4, we know that if a tower is formed it is composed only of two robots. (i) Besides, by Lemma 9.3 once a tower is formed, the robots of this tower consider two opposite global directions. This implies that, during the whole execution, x robots consider the clockwise direction and y robots consider the counter-clockwise direction.

While executing $\mathbb{PEF}\_3+$, a robot considers a direction (right or left) at each round. An isolated robot does not change the direction it considers. Therefore, by (i) and knowing that all the edges of a $\mathcal{ST}$ ring are always present, the cover time is equal to nb rounds, where nb is the maximal number of edges in the direction d separating two robots considering the global direction d in a configuration γ. The value nb is maximal when all the robots that consider the same global direction in γ are on consecutive nodes in γ. In this case nb is equal to $n - x + 1$ for the robots that consider the clockwise direction in γ and to $n - y + 1$ for the robots that consider the counter-clockwise direction in γ. Since $x + y = \mathcal{R}$, to maximize nb, x and y must be two close values. Hence, the cover time of $\mathbb{PEF}\_3+$ is equal to $n - (\mathcal{R}/2) + 1$ rounds. □

We now prove that the cover time of $\mathbb{PEF}\_3+$ in $\mathcal{ST}$ rings is moreover asymptotically optimal when executed in $\mathcal{ST}$ rings (see Lemma 9.10). To do so, we show that the cover time of any deterministic algorithm solving the perpetual exploration problem in $\mathcal{ST}$ rings of size n using $\mathcal{R}$ robots is equal to the cover time of our algorithm (proved in Lemma 9.9).

Lemma 9.10. *Any deterministic algorithm under* $\mathcal{M}_{exploration}$ *using* $\mathcal{R}$ *robots, satisfying the perpetual exploration problem in* $\mathcal{ST}$ *rings of size* n *has a cover time in* $\Omega(n - \mathcal{R})$ *rounds.*

Proof. Assume, by contradiction, that there exists a deterministic algorithm $\mathcal{A}$ using $\mathcal{R}$ robots, solving the perpetual exploration problem in $\mathcal{ST}$ rings of size n, that possesses a cover time less or equals to $\tau = \lceil (n - \mathcal{R})/2 \rceil - 1$ rounds.

Consider the execution of $\mathcal{A}$ in a $\mathcal{ST}$ ring of size n such that initially the robots are on distinct positions, but all are located on consecutive nodes. Call this initial configuration a block, and denote it γ.

Call x one of the nodes at the extremity of the block. Call y the node at distance $\lceil (n - \mathcal{R})/2 \rceil$ of x such that there is no robot located on the nodes between x and y.

Since each robot can cross at most one edge at each round, this implies that each robot can explore at most $\lceil (n - \mathcal{R})/2 \rceil - 1$ different nodes in τ rounds during the execution of $\mathcal{A}$ starting from γ. Note that all the nodes visited by a robot are consecutive nodes. Therefore, whatever the direction considered by each of the robots, none of them succeed to visit the node y during the τ first rounds of the execution of $\mathcal{A}$ starting from γ. Hence there is a contradiction with the fact that $\mathcal{A}$ is deterministic algorithm using $\mathcal{R}$ robots, solving the perpetual exploration problem in $\mathcal{ST}$ rings of size n, and having a cover time of τ rounds.

This implies that there is no deterministic algorithm solving the perpetual exploration problem in $\mathcal{ST}$ rings of size n, using $\mathcal{R}$ robots and having a cover time less or equals to $\lceil (n - \mathcal{R})/2 \rceil - 1$ rounds. Hence, since $\lceil (n - \mathcal{R})/2 \rceil - 1$ is in $\Omega(n - \mathcal{R})$, the lemma is proved. □

To conclude, as explained at the beginning of this section, and thanks to the two previous lemmas, we can deduce the following theorem.

Theorem 9.2. $\mathbb{PEF}\_3+$ *is a speculative algorithm under* $\mathcal{M}_{exploration}$ *for the perpetual exploration problem with respect to* $\mathcal{ST}$ *and* $\mathcal{COT}$*, in function of the cover time in rounds, and its cover time in* $\mathcal{ST}$ *rings is asymptotically optimal.*

9.3 With Two Robots

In this section, we study the perpetual exploration of rings of any size with two robots. We first prove that two robots are not able to perpetually explore $\mathcal{COT}$ rings of size strictly greater than three (refer to Theorem 9.3). Then, we provide $\mathbb{PEF}\_2$ (see Theorem 9.4), an algorithm using two robots that solves the perpetual exploration in the remaining case, i.e., $\mathcal{COT}$ rings of size three.

9.3.1 $\mathcal{COT}$ Rings of Size 4 or More

The proof of our impossibility result presented in Theorem 9.3 is based on the construction of an adequate sequence of evolving graphs and the application of the generic framework proposed in [34] (see Section 4.2 for more information about this framework).

In order to build the evolving graphs sequence suitable for the proof of our impossibility result, we need the following technical lemma.

Lemma 9.11. *Any execution of a deterministic perpetual exploration algorithm in $\mathcal{COT}$ rings of size 4 or more using 2 robots satisfies: for any robot state s, there exists a time $t' \geq 0$ such that, if, from time 0 to a time t, the robots have not explored the whole ring, have not formed a tower, each robot has only visited at most two adjacent nodes, a robot r is located, in state s at time t, on a node u satisfying $OneEdge(u, t, t + t')$, then at least one robot on u leaves u at time $t + t'$.*

Proof. By contradiction, assume that there exists an execution ε of a deterministic perpetual exploration algorithm $\mathcal{A}$ in a $\mathcal{COT}$ ring $\mathcal{G} = \{G_0, G_1, \ldots\}$ (with for any $i \in \mathbb{N}, G_i = (V, E_i)$) of size 4 or more using 2 robots r_1 and r_2 satisfying: it exists a state s such that, for any integer t' ($t' \geq 0$), from time 0 to a time t, (i) the exploration of the whole ring has not been done yet; (ii) none of the robots have formed a tower; (iii) at time t each robot has only visited at most two adjacent nodes of $\mathcal{G}$; and (iv) at time t one of the robots (without lost of generality, r_1) is located, in state s, on a node u of $\mathcal{G}$ satisfying $OneEdge(u, t, t + t')$ and neither r_1 nor r_2 leave u at time $t + t'$.

Let $\mathcal{R}$ be the set of nodes of $\mathcal{G}$ visited by r_1 from time 0 to time t in ε. Note that, at time t, as each robot has only visited at most two adjacent nodes, then $1 \leq |\mathcal{R}| \leq 2$. Let i (resp. f) be the node in $\mathcal{G}$ where r_1 is located at time 0 (resp. t). If $|\mathcal{R}| = 2$, let a be the node of $\mathcal{R}$ such that $a \neq i$, otherwise (i.e., $|\mathcal{R}| = 1$) let $a = i$. By assumption, either $f = i$ or f is an adjacent node of i and in this later case $a = f$.

We construct a $\mathcal{COT}$ ring $\mathcal{G}' = \{G'_0, G'_1, \ldots\}$ (with $G'_i = (V', E'_i)$ for any $i \in \mathbb{N}$) such that the footprint of $\mathcal{G}'$, $G' = (V', E')$, contains 8 nodes in the following way. Let i'_1 be an arbitrary node of $\mathcal{G}'$. Let us construct nodes i'_2, a'_1, a'_2, f'_1, and f'_2 of $\mathcal{G}'$ in function of i'_1 and of nodes i, a, and f of $\mathcal{G}$ as explained in Figure 9.1 (in which we represent a part of the footprints of $\mathcal{G}$ and $\mathcal{G}'$). Note that this construction ensures that f'_1 and f'_2 are adjacent in $\mathcal{G}'$ in any case. We recall that $d(x, y)$ corresponds to the distance in G' between the nodes x and y of $\mathcal{G}$.

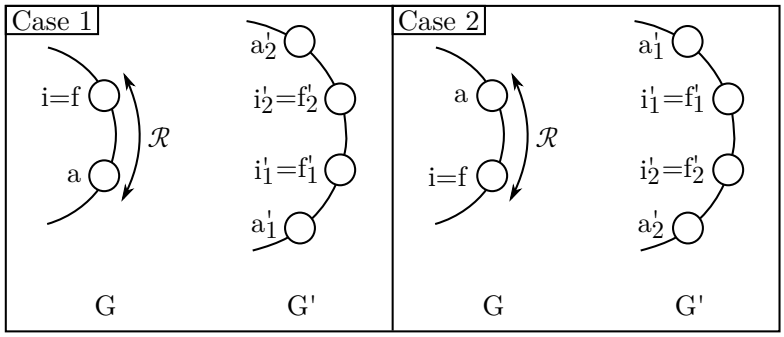
We denote by $r(k)$ (resp. $l(k)$) the adjacent edge in the clockwise (resp. counter-clockwise) direction of a node k. For any $j \in \{0, \ldots, t-1\}$, let E'_j be the set E' with the following set of additional constraints[1]:

$$\begin{cases} r(i'_1) \in E'_j \text{ and } l(i'_2) \in E'_j & \text{iff } r(i) \in E_j \\ l(i'_1) \in E'_j \text{ and } r(i'_2) \in E'_j & \text{iff } l(i) \in E_j \\ r(a'_1) \in E'_j \text{ and } l(a'_2) \in E'_j & \text{iff } r(a) \in E_j \\ l(a'_1) \in E'_j \text{ and } r(a'_2) \in E'_j & \text{iff } l(a) \in E_j \end{cases}$$

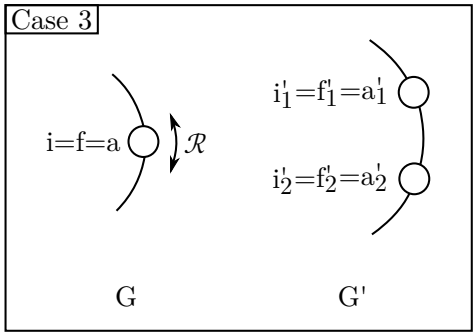
For any $j \geq t$, let E'_j be the set $E' \setminus \{(f'_1, f'_2)\}$.

[1]Note that the construction of i'_1, i'_2, a'_1, a'_2, f'_1, and f'_2 ensures us that there is no contradiction between these constraints in all cases.

• If i=f and d(i, a)=1:



• If i=f and d(i, a)=0:



• If i≠f and d(i, a)=1:

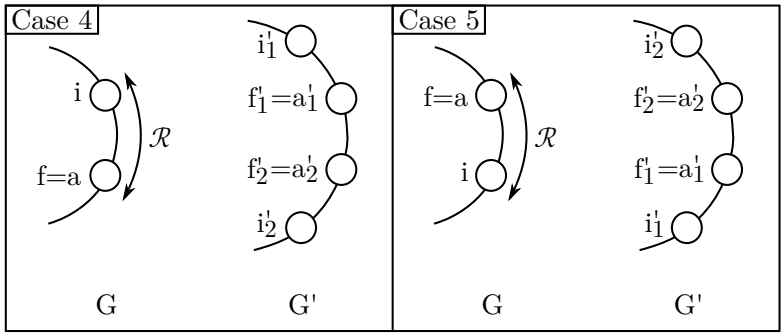


Figure 9.1: Construction of the nodes of G' in function of the nodes of G.

Now, we consider the execution ε' of $\mathcal{A}$ in $\mathcal{G}'$ starting from the configuration where r_1 (resp. r_2) is on node i'_1 (resp. on node i'_2) such that the two robots have opposite chirality and that r_1 has the same chirality as in ε. The execution ε' satisfies the following set of claims.

Claim 1: Until time t, $\mathbf{r_1}$ and $\mathbf{r_2}$ execute the same actions in a symmetrical way in ε'.

Consider that, during the Look phase of a time j, the two robots have the same view in ε'. The two robots have not the same chirality and $\mathcal{A}$ is deterministic, then, during the Move phase of time j, they are executing the same action in a symmetrical way (either not move or move in opposite directions). This implies that, at time $j+1$, r_1 and r_2 have again the same state.

There are only two robots executing $\mathcal{A}$ in $\mathcal{G}'$. Hence, if a tower is formed, it is composed of r_1 and r_2. If from time 0 to time t, the robots are executing the same actions in a symmetrical way, then, by construction of $\mathcal{G}'$ and by the way we initially placed r_1 and r_2 in ε', the two robots see the same local environment at each instant time in $\{0, \ldots, t\}$.

At time 0, by construction of $\mathcal{G}'$ and by the way we placed r_1 and r_2 in ε', the two robots have the same view.

By recurrence and using the arguments of the two first paragraphs, we conclude that, from time 0 to time t, r_1 and r_2 execute the same actions in a symmetrical way in ε'.

Claim 2: Until time t, $\mathbf{r_1}$ and $\mathbf{r_2}$ never form a tower in ε'.

By construction of ε', the two robots are initially at an odd distance. By Claim 1, at a time $0 < j + 1 < t$, the two robots are either at the same distance, at a distance increased of 2, or at a distance decreased of 2 with respect to their distance at time j. Moreover, since $\mathcal{G}'$ possesses an even number of edges, this implies that, until time t, the robots are always at an odd distance from each other.

Claim 3: Until time t, $\mathbf{r_1}$ executes in ε' the same sequence of actions as in ε.

Consider that, during the Look phase of a time j, r_1 has the same view in ε and in ε'. As $\mathcal{A}$ is deterministic, then, during the Move phase of time j, r_1 executes the same action (either not move, or move in the same direction) in ε and in ε'. This implies that, during the Look phase of time $j + 1$, r_1 possesses the same state in ε and in ε'.

By assumption, until time t, there is no tower in ε. By Claim 2, there is no tower in ε' until time t. Hence, in the case where r_1 executes the same actions in ε and in ε' from time 0 to time t, r_1 sees the same local environment in ε and in ε' until time t (by construction of $\mathcal{G}'$ and the initial location of r_1 in ε').

At time 0, r_1 has the same view in ε and in ε' (by construction of $\mathcal{G}'$ and the initial location of r_1 in ε').

By recurrence and using the arguments of the two first paragraphs, we conclude that, from time 0 to time t, r_1 executes the same actions in ε and in ε'.

Claim 4: At time t, $\mathbf{r_1}$ and $\mathbf{r_2}$ are on two adjacent nodes in ε' and are both in state s.

By Claims 1 and 3, and by construction of $\mathcal{G}'$, we know that at time t, r_1 is on node f'_1 while r_2 is on node f'_2. These nodes are adjacent by construction of $\mathcal{G}'$.

By Claim 1, as r_1 and r_2 have opposite chirality, they have the same state at time t in ε'. By Claim 3, r_1 is in the same state at time t in ε and in ε'. Since r_1 is in state s at time t in ε by assumption, we have the claim.

By construction of $\mathcal{G}'$, f'_1 (resp. f'_2) satisfies the property $OneEdge(f'_1, t, +\infty)$ (resp. $OneEdge(f'_2, t, +\infty)$). Then, by assumption, r_1 (resp. r_2) does not leave node f'_1 (resp. f'_2) after time t. As $\mathcal{G}'$ counts 8 nodes, we obtain a contradiction with the fact that $\mathcal{A}$ is a deterministic algorithm solving the perpetual exploration problem for $\mathcal{COT}$ rings using two robots. □

Theorem 9.3. *There exists no deterministic algorithm satisfying the perpetual exploration problem under $\mathcal{M}_{exploration}$ in $\mathcal{COT}$ rings of size 4 or more using two robots.*

Proof. By contradiction, assume that there exists a deterministic algorithm $\mathcal{A}$ satisfying the perpetual exploration specification in a deterministic way in any $\mathcal{COT}$ ring of size 4 or more using two fully-synchronous robots r_1 and r_2.

Consider the $\mathcal{COT}$ graph $\mathcal{G} = \{G_0, G_1, \ldots\}$ whose footprint G is a ring of size 4 or more and such that $\forall i \in \mathbb{N}, G_i = G$.

Consider three nodes u, v, and w of $\mathcal{G}$, such that node v is the adjacent node of u in the clockwise direction, and node w is the adjacent node of v in the clockwise direction. For any node x of $\mathcal{G}$ denote respectively by e_{xr} and e_{xl} the clockwise and the counter-clockwise adjacent edges of x. Note that $e_{ur} = e_{vl}$, and $e_{vr} = e_{wl}$.

$\mathcal{A}$ is a deterministic algorithm solving the perpetual exploration problem in $\mathcal{COT}$ rings of size 4 or more using 2 robots, hence, by Lemma 9.11, any execution of $\mathcal{A}$ satisfies: for any robot state s, there exists a time $t' \geq 0$ such that, if, from time 0 to a time t, the robots have not explored the whole ring, have not formed a tower, each robot has only visited at most two adjacent nodes, a robot is located, in state s at time t, on a node x satisfying $OneEdge(x, t, t + t')$ then at least a robot located on node x leaves x at time $t + t'$. Let $\mathcal{T}_{\mathcal{A}}$ be the largest integer t' whatever the

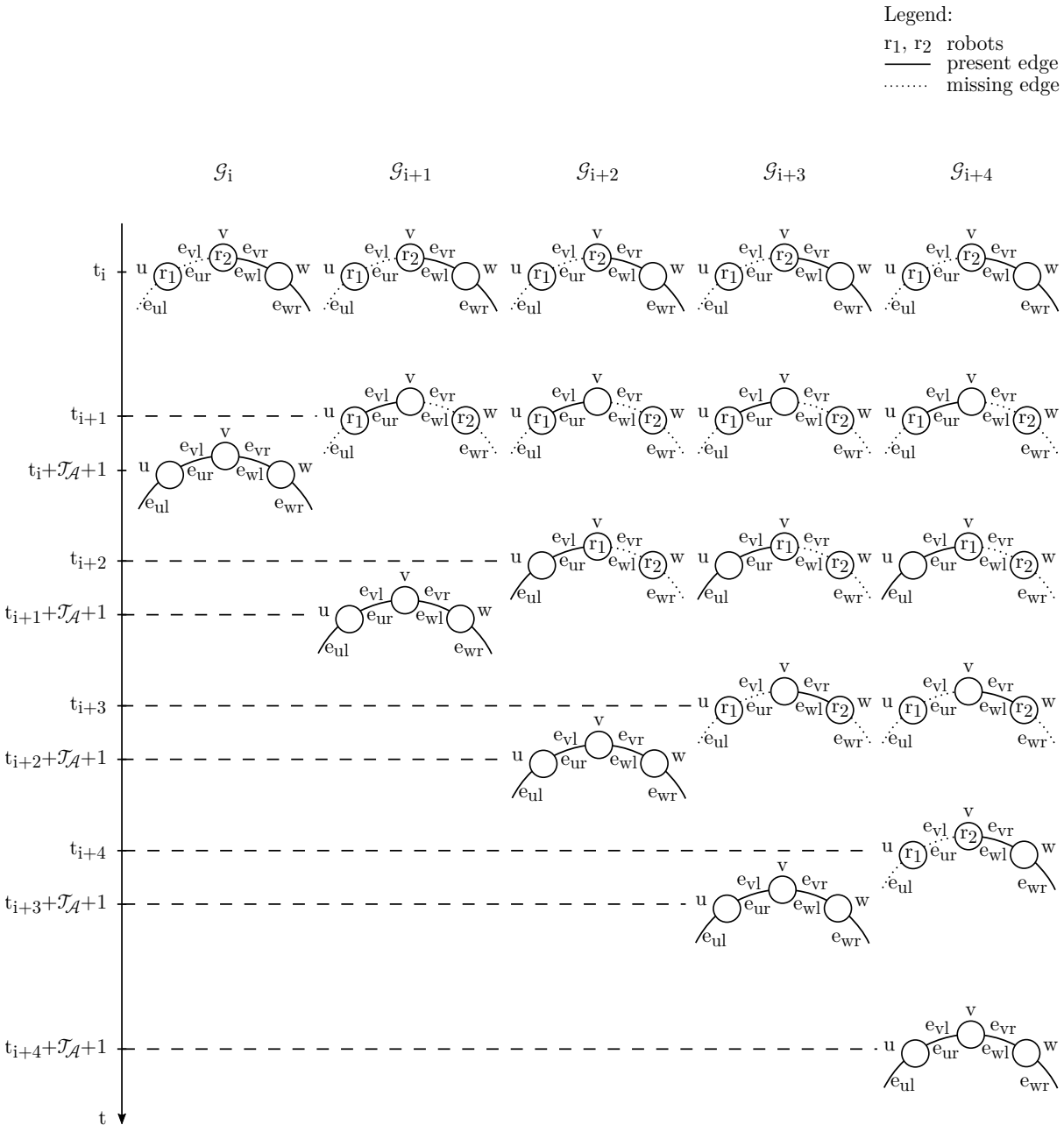


Figure 9.2: Construction of $\mathcal{G}_j$ and evolution of the robots in $\mathcal{G}_j$ until time t_j, for $j = \{i, i+1, i+2, i+3, i+4\}$.

states considered by the robots executing $\mathcal{A}$. Since the states reached by the robots executing $\mathcal{A}$ are among a finite set of values, $\mathcal{T}_\mathcal{A}$ exists and is bounded.

Let $\mathcal{G}'$ be $\mathcal{G}\backslash\{(\{e_{ul}, e_{ur}\}, \mathcal{T}_\mathcal{A})\}$. Let ε be the execution of $\mathcal{A}$ in $\mathcal{G}'$ starting from the configuration where r_1 (resp. r_2) is located on node u (resp. v).

Our goal is to construct a sequence of $\mathcal{COT}$ rings denoted $(\mathcal{G}_m)_{m\in\mathbb{N}}$ such that $\mathcal{G}_0 = \mathcal{G}'$ and, for any $i \geq 0$, (i) $\mathcal{G}_i$ exists for a given $i \in \mathbb{N}$ multiple of 4; (ii) until time t_i, the two robots never form a tower in ε_i ($(t_m)_{m\in\mathbb{N}}$ being a strictly increasing sequence with $t_0 = 0$, and ε_i being the execution of $\mathcal{A}$ in $\mathcal{G}_i$ starting from the same configuration as ε); (iii) until time t_i, r_1 (resp. r_2) has only visited at most two nodes among $\{u, v\}$ (resp. $\{v, w\}$) in ε_i; and (iv) at time t_i, r_1 (resp. r_2) is located on node v (resp. w). First, we show in the next paragraph (items 1 to 8) that, if some such $\mathcal{G}_i$ exists and, moreover, ensures the existence of an integer $t'_i \geq 0$ such that until time $t_i + t'_i + 1$ the conditions (ii) and (iii) are satisfied in ε_i, then we can construct $\mathcal{G}_{i+1}$, $\mathcal{G}_{i+2}$, $\mathcal{G}_{i+3}$

and $\mathcal{G}_{i+4}$ as shown in Figure 9.2. We prove, after that, that our construction implies the well definition of $(\mathcal{G}_m)_{m\in\mathbb{N}}$.

1. If, at time t_i, the configuration γ_i is such that the robot r_1 is on node u and has none of its adjacent edge present while r_2 is on node v and has only its adjacent edge e_{vr} present in ε_i, then, since $\mathcal{G}_i$ is a $\mathcal{COT}$ ring and satisfies the conditions (ii) and (iii) until time t_i, we use Lemma 9.11 to exhibit the smallest time $t'_i \geq 0$ such that, if the configuration γ_i lasts from time t_i to time $t_i + t'_i$, then r_2 moves at time $t_i + t'_i$. By definition $t'_i \leq \mathcal{T}_{\mathcal{A}}$. In ε_i, at time $t_i + t'_i + 1$, r_2 is on node w (since only its adjacent edge e_{vr} is present at time $t_i + t'_i$) and r_1 is still on node u. Therefore, the conditions (ii) and (iii) are still satisfied in ε_i until time $t_i + t'_i + 1$.

   We then define $\mathcal{G}_{i+1}$ such that $\mathcal{G}_{i+1}$ and $\mathcal{G}_i$ have the same footprint and $\mathcal{G}_{i+1} = \mathcal{G}_i \otimes_{t_i+t'_i} (\mathcal{G}\backslash\{(\{e_{ul}, e_{wl}, e_{wr}\}, \{t_i + t'_i + 1, \ldots, t_i + t'_i + 1 + \mathcal{T}_{\mathcal{A}}\})\})$. Note that $\mathcal{G}_{i+1}$ is a $\mathcal{COT}$ ring (since it is indistinguishable from $\mathcal{G}$ after time $t_i + t'_i + 1 + \mathcal{T}_{\mathcal{A}}$).

   $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ are indistinguishable for robots until time $t_i + t'_i$. This implies that, at time $t_i + t'_i + 1$, r_1 (resp. r_2) is on node u (resp. w) in ε_{i+1}. Moreover, the conditions (ii) and (iii) are satisfied in ε_{i+1} until time $t_i + t'_i + 1$.

2. Let $t_{i+1} = t_i + t'_i + 1$.

3. Using similar arguments as in Item 1, if, at time t_{i+1}, the configuration γ_{i+1} is such that the robot r_2 is on node w and has none of its adjacent edge present while r_1 is on node u and has only its adjacent edge e_{ur} present in ε_{i+1}, we prove that there exists a time $\mathcal{T}_{\mathcal{A}} \geq t'_{i+1} \geq 0$ such that, if the configuration γ_{i+1} lasts until time $t_{i+1} + t'_{i+1}$, then, at time $t_{i+1} + t'_{i+1} + 1$, r_1 is on node v, r_2 is on node w and the conditions (ii) and (iii) are satisfied in ε_{i+1} until time $t_{i+1} + t'_{i+1} + 1$.

   We then define $\mathcal{G}_{i+2}$ such that $\mathcal{G}_{i+2}$ and $\mathcal{G}_i$ have the same footprint and $\mathcal{G}_{i+2} = \mathcal{G}_{i+1} \otimes_{t_{i+1}+t'_{i+1}} (\mathcal{G}\backslash\{(\{e_{wl}, e_{wr}\}, \{t_{i+1} + t'_{i+1} + 1, \ldots, t_{i+1} + t'_{i+1} + 1 + \mathcal{T}_{\mathcal{A}}\})\})$. Note that $\mathcal{G}_{i+2}$ is a $\mathcal{COT}$ ring (since it is indistinguishable from $\mathcal{G}$ after time $t_{i+1} + t'_{i+1} + 1 + \mathcal{T}_{\mathcal{A}}$).

   $\mathcal{G}_{i+1}$ and $\mathcal{G}_{i+2}$ are indistinguishable for robots until time $t_{i+1} + t'_{i+1}$. This implies that, at time $t_{i+1} + t'_{i+1} + 1$, r_1 (resp. r_2) is on node v (resp. w) in ε_{i+2}. Moreover, the conditions (ii) and (iii) are satisfied in ε_{i+2} until time $t_{i+1} + t'_{i+1} + 1$.

4. Let $t_{i+2} = t_{i+1} + t'_{i+1} + 1$.

5. Using similar arguments as in Item 1, if, at time t_{i+2}, the configuration γ_{i+2} is such that the robot r_2 is on node w and has none of its adjacent edge present while r_1 is on node v and has only its adjacent edge e_{vl} present in ε_{i+2}, we prove that there exists a time $\mathcal{T}_{\mathcal{A}} \geq t'_{i+2} \geq 0$ such that, if the configuration γ_{i+2} lasts until time $t_{i+2} + t'_{i+2}$, then, at time $t_{i+2} + t'_{i+2} + 1$, r_1 is on node u, r_2 is on node w and the conditions (ii) and (iii) are satisfied in ε_{i+2} until time $t_{i+2} + t'_{i+2} + 1$.

   We then define $\mathcal{G}_{i+3}$ such that $\mathcal{G}_{i+3}$ and $\mathcal{G}_i$ have the same footprint and $\mathcal{G}_{i+3} = \mathcal{G}_{i+2} \otimes_{t_{i+2}+t'_{i+2}} (\mathcal{G}\backslash\{(\{e_{ul}, e_{ur}, e_{wr}\}, \{t_{i+2} + t'_{i+2} + 1, \ldots, t_{i+2} + t'_{i+2} + 1 + \mathcal{T}_{\mathcal{A}}\})\})$. Note that $\mathcal{G}_{i+3}$ is a $\mathcal{COT}$ ring (since it is indistinguishable from $\mathcal{G}$ after time $t_{i+2} + t'_{i+2} + 1 + \mathcal{T}_{\mathcal{A}}$).

   $\mathcal{G}_{i+2}$ and $\mathcal{G}_{i+3}$ are indistinguishable for robots until time $t_{i+2} + t'_{i+2}$. This implies that, at time $t_{i+2} + t'_{i+2} + 1$, r_1 (resp. r_2) is on node u (resp. w) in ε_{i+3}. Moreover, the conditions (ii) and (iii) are satisfied in ε_{i+3} until time $t_{i+2} + t'_{i+2} + 1$.

6. Let $t_{i+3} = t_{i+2} + t'_{i+2} + 1$.

7. Using similar arguments as in Item 1, if, at time t_{i+3}, the configuration γ_{i+3} is such that the robot r_1 is on node u and has none of its adjacent edge present while r_2 is on node w and has

only its adjacent edge e_{wl} present in ε_{i+3}, we prove that there exists a time $\mathcal{T}_{\mathcal{A}} \geq t'_{i+3} \geq 0$ such that, if the configuration γ_{i+3} lasts until time $t_{i+3} + t'_{i+3}$, then, at time $t_{i+3} + t'_{i+3} + 1$, r_1 is on node u, r_2 is on node v and the conditions (ii) and (iii) are satisfied in ε_{i+3} until time $t_{i+3} + t'_{i+3} + 1$.

We then define $\mathcal{G}_{i+4}$ such that $\mathcal{G}_{i+4}$ and $\mathcal{G}_i$ have the same footprint and $\mathcal{G}_{i+4} = \mathcal{G}_{i+3} \otimes_{t_{i+3}+t'_{i+3}} (\mathcal{G} \backslash \{(\{e_{ul}, e_{ur}\}, \{t_{i+3} + t'_{i+3} + 1, \ldots, t_{i+3} + t'_{i+3} + 1 + \mathcal{T}_{\mathcal{A}}\})\})$. Note that $\mathcal{G}_{i+4}$ is a $\mathcal{COT}$ ring (since it is indistinguishable from $\mathcal{G}$ after time $t_{i+3} + t'_{i+3} + 1 + \mathcal{T}_{\mathcal{A}}$).

$\mathcal{G}_{i+3}$ and $\mathcal{G}_{i+4}$ are indistinguishable for robots until time $t_{i+3} + t'_{i+3}$. This implies that, at time $t_{i+3} + t'_{i+3} + 1$, r_1 (resp. r_2) is on node u (resp. w) in ε_{i+4}. Moreover, the conditions (ii) and (iii) are satisfied in ε_{i+4} until time $t_{i+3} + t'_{i+3} + 1$.

8. Let $t_{i+4} = t_{i+3} + t'_{i+3} + 1$.

Note that $\mathcal{G}_0$ trivially satisfies assumptions (i) to (iv) for $t_0 = 0$. Also, given a $\mathcal{G}_i$ with $i \in \mathbb{N}$ multiple of 4, $\mathcal{G}_{i+4}$ exists and we proved that it satisfies assumptions (ii) to (iv). In other words, $(\mathcal{G}_n)_{n\in\mathbb{N}}$ is well-defined.

We define the evolving graph $\mathcal{G}_\omega$ such that $\mathcal{G}_\omega$ and $\mathcal{G}_0$ have the same footprint, and for all $i \in \mathbb{N}$, $\mathcal{G}_\omega$ shares a common prefix with $\mathcal{G}_i$ until time $t_i + t'_i$. As the sequence $(t_m + t'_m)_{m\in\mathbb{N}}$ is increasing by construction, this implies that the sequence $(\mathcal{G}_m)_{m\in\mathbb{N}}$ converges to $\mathcal{G}_\omega$.

Note that, for any edge of $\mathcal{G}_\omega$, the intervals of times where this edge is absent (if any) are finite and disjoint. This implies that all the edges of $\mathcal{G}_\omega$ are infinitely often present. Therefore, $\mathcal{G}_\omega$ is a $\mathcal{COT}$ ring.

Applying the theorem of [34], we obtain that, until time $t_i + t'_i$, the execution of $\mathcal{A}$ in $\mathcal{G}_\omega$ is identical to the one in $\mathcal{G}_i$. This implies that, executing $\mathcal{A}$ in $\mathcal{G}_\omega$ (whose footprint is a ring of size 4 or more), r_1 and r_2 only visit the nodes among $\{v, w, x\}$. This is contradictory with the fact that $\mathcal{A}$ satisfies the perpetual exploration specification in $\mathcal{COT}$ rings of size 4 or more using 2 robots. □

9.3.2 $\mathcal{COT}$ Rings of Size 3

In this section, we present $\mathbb{PEF}\_2$, a deterministic algorithm solving the perpetual exploration in $\mathcal{COT}$ rings of size 3 with two robots.

This algorithm works as follows. Each robot disposes only of its *dir* variable. If at a time t, a robot is isolated on a node with only one adjacent edge, then it points to this edge. Otherwise (i.e., none of the adjacent edge is present, both adjacent edges are present, or the other robot is present on the same node), the robot keeps its current direction.

Algorithm 9.2 $\mathbb{PEF}\_2$

1: **Robots** :: $!ExistsOtherRobotsOnNode() \wedge ExistsEdge(\overline{dir}, current) \wedge !ExistsEdge(dir, current)$
$\longrightarrow dir := \overline{dir}$

Theorem 9.4. *$\mathbb{PEF}\_2$ satisfies the perpetual exploration problem under $\mathcal{M}_{exploration}$ in $\mathcal{COT}$ rings of 3 nodes using 2 robots.*

Proof. Consider any execution of $\mathbb{PEF}\_2$ in any $\mathcal{COT}$ ring of size 3 with 2 robots. By the connected-over-time assumption, each node has at least one adjacent edge infinitely often present. This implies that any tower is broken in finite time (as robots meet only when they consider opposite directions and move as soon as it is possible). Two cases are now possible.

Case 1: There exists infinitely often a tower in the execution.

Note that, if a tower is formed at a time t, then the three nodes have been visited between time $t - 1$ and time t. Then, the three nodes are infinitely often visited by a robot in this case.

Case 2: There exists a time t after which the robots are always isolated.

By contradiction, assume that there exists a time t' such that a node u is never visited after t'. As the ring has 3 nodes, that implies that, after time $max\{t, t'\}$, either the robots are always switching their position or they stay on their respective nodes.

In the first case, during the Look phase of each time greater than $max\{t, t'\}$, the respective variables *dir* of the two robots contain the direction leading to u (since it previously moves in this direction). As at least one of the adjacent edges of u is infinitely often present, a robot crosses it in a finite time, that is contradictory with the fact that u is not visited after t'.

The second case implies that both adjacent edges to the location of both robots are always absent after time t (since a robot moves as soon as it is possible), that is contradictory with the connected-over-time assumption.

In both cases, $\mathbb{PEF}\_2$ satisfies the perpetual exploration specification.

□

9.4 With One Robot

This section leads a similar study as the one in Section 9.3 but in the case of the perpetual exploration of rings of any size with a single robot. In this section we prove a negative result since Theorem 9.5 states that a single robot is not able to perpetually explore $\mathcal{COT}$ rings of any size (i.e., of size 3 or more).

Similarly to the previous section, the proof of our impossibility result presented in Theorem 9.5 is based on the construction of an adequate sequence of evolving graphs and the application of the generic framework proposed in [34] (see Section 4.2 for more information about this framework).

In order to build the evolving graphs sequence suitable for the proof of our impossibility result, we need the following trivial lemma that states that if the robot has only one adjacent present edge during a long time then, in finite time, it will move. Trivially, if this was not the case and if the adjacent missing edge of the robot is actually an eventual missing edge, then the perpetual exploration cannot be solved by the robot.

Lemma 9.12. *Any execution of a perpetual exploration algorithm in $\mathcal{COT}$ rings of size 3 or more using one robot r satisfies: for any robot state s, there exists a time $t' \geq 0$ such that if r is located on a node u, in state s at a time t, and satisfying $OneEdge(u, t, t + t')$, then r leaves u at time $t + t'$.*

Theorem 9.5. *There exists no deterministic algorithm satisfying the perpetual exploration problem under $\mathcal{M}_{exploration}$ in $\mathcal{COT}$ rings of size 3 or more using a single robot.*

Proof. By contradiction, assume that there exists a deterministic algorithm $\mathcal{A}$ satisfying the perpetual exploration specification in any $\mathcal{COT}$ ring of size 3 or more using a single robot r.

Consider the $\mathcal{COT}$ graph $\mathcal{G} = \{G_0, G_1, \ldots\}$ whose footprint G is a ring of size 3 or more and such that $\forall i \in \mathbb{N}, G_i = G$.

Let u be a node of $\mathcal{G}$ and let v be the adjacent node of u in the clockwise direction. For any node x of $\mathcal{G}$ denote respectively by e_{xr} and e_{xl} the clockwise and the counter-clockwise adjacent edges of x.

Let $\mathcal{G}'$ be $\mathcal{G}\backslash\{(\{e_{ur}\}, \mathbb{N})\}$. Let ε be the execution of $\mathcal{A}$ on $\mathcal{G}'$ starting from the configuration where r is located on node u.

Our goal is to construct a sequence of $\mathcal{COT}$ rings denoted $(\mathcal{G}_m)_{m\in\mathbb{N}}$ such that $\mathcal{G}_0 = \mathcal{G}'$ and, for any $i \geq 0$, only nodes among $\{u, v\}$ have been visited until time t_i in ε_i (the execution of $\mathcal{A}$ in

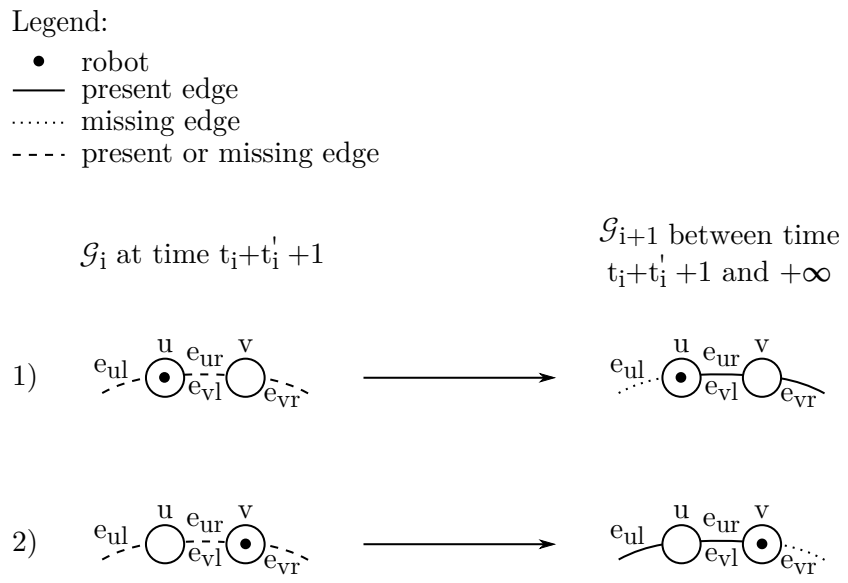


Figure 9.3: Construction of $\mathcal{G}_{i+1}$ from $\mathcal{G}_i$.

$\mathcal{G}_i$ starting from the same configuration as ε), $(t_m)_{m\in\mathbb{N}}$ being a strictly increasing sequence with $t_0 = 0$. First, we show in the next paragraph that, if some such $\mathcal{G}_i$ exists and, moreover, ensures the existence of an integer $t'_i \geq 0$ such that until time $t_i + t'_i + 1$ only nodes among $\{u, v\}$ have been visited in ε_i, then we can construct $\mathcal{G}_{i+1}$ as shown in Figure 9.3. We prove, after that, that our construction guarantees the existence of such a t'_i, implying the well definition of $(\mathcal{G}_m)_{m\in\mathbb{N}}$.

Since $\mathcal{G}_i$ is a $\mathcal{COT}$ ring, and since $\mathcal{A}$ is a deterministic algorithm solving the perpetual exploration problem in $\mathcal{COT}$ rings of size 3 or more using a single robot, when the configuration γ_i at time t_i is such that there is exactly one adjacent edge present to the location of r, we use Lemma 9.12 to exhibit the smallest integer $t'_i \geq 0$ such that if the configuration γ_i last from time t_i to time $t_i + t'_i$, then r moves at time $t_i + t'_i$.

In the following we show how we construct the dynamic graph $\mathcal{G}_{i+1}$ in function of $\mathcal{G}_i$, t_i and t'_i. As we assume that in $\mathcal{G}_i$, until time $t_i + t'_i + 1$, only nodes among $\{u, v\}$ have been visited, then the following cases are possible.

1. if, at time $t_i + t'_i + 1$ in $\mathcal{G}_i$, r is on node v then let $E_i = \{e_{vr}\}$.

2. if, at time $t_i + t'_i + 1$ in $\mathcal{G}_i$, r is on node u then let $E_i = \{e_{ul}\}$.

For each case, we define $\mathcal{G}_{i+1}$ such that $\mathcal{G}_{i+1}$ and $\mathcal{G}_i$ have the same footprint and $\mathcal{G}_{i+1} = \mathcal{G}_i \otimes_{t_i+t'_i} (\mathcal{G}\backslash\{(E_i, \{t_i + t'_i + 1, \ldots, +\infty\})\})$.

$\mathcal{G}_{i+1}$ is a $\mathcal{COT}$ ring since it possesses only one eventual missing edge. Note that $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ are indistinguishable for r until time $t_i + t'_i$. This implies that, at time $t_i + t'_i$, r is on the same node in ε_i and in ε_{i+1}. By construction of t'_i, r moves at time $t_i + t'_i$ in ε_{i+1}. Moreover, even if r moves during the Move phase of time $t_i+t'_i$, at time $t_i+t'_i+1$, it is still on nodes among $\{u, v\}$, by assumption on $\mathcal{G}_i$ and construction of $\mathcal{G}_{i+1}$, since from time t_i to time $t_i + t'_i$ the edges permitting r to go on a node other than the nodes among $\{u, v\}$ are missing.

Let $t_{i+1} = t_i + t'_i + 1$. Then we can construct recursively each dynamic ring of $(\mathcal{G}_m)_{m\in\mathbb{N}}$ by applying the argues above on all the possible configurations reached by the movement of r at time $t_{i+1} + t'_{i+1} + 1$ on $\mathcal{G}_{i+1}$.

Note that the recurrence can be initiated, since $\mathcal{G}_0$ exists, is a $\mathcal{COT}$ ring and from time $t_0 = 0$ to time $t_0 + t'_0 + 1$ only nodes among $\{u, v\}$ have been visited. In other words, $(\mathcal{G}_m)_{m\in\mathbb{N}}$ is well-defined.

We can then define the evolving graph $\mathcal{G}_\omega$ such that $\mathcal{G}_\omega$ and $\mathcal{G}_0$ have the same footprint and for all $i \in \mathbb{N}$, $\mathcal{G}_\omega$ shares a common prefix with $\mathcal{G}_i$ until time $t_i + t'_i$. As the sequence $(t_m + t'_m)_{m\in\mathbb{N}}$ is increasing by construction, this implies that the sequence $(\mathcal{G}_m)_{m\in\mathbb{N}}$ converges to $\mathcal{G}_\omega$.

Note that $\mathcal{G}_\omega$ is a $\mathcal{COT}$ ring. Indeed, there is at most one eventual missing edge in $\mathcal{G}_\omega$. Applying the theorem of [34], we obtain that, until time $t_i + t'_i$, the execution of $\mathcal{A}$ on $\mathcal{G}_\omega$ is identical to the one on $\mathcal{G}_i$. This implies that, executing $\mathcal{A}$ on $\mathcal{G}_\omega$ (whose footprint is a ring of size 3 or more), r only visits the nodes among $\{u, v\}$. This is contradictory with the fact that $\mathcal{A}$ satisfies the perpetual exploration specification on $\mathcal{COT}$ rings of size 3 or more using a single robot. □

9.5 Summary

We analyzed the computability of the perpetual exploration problem in highly dynamic rings (i.e., in $\mathcal{COT}$ rings). We proved that three (resp., two) robots with very few capacities are necessary to solve the perpetual exploration problem in $\mathcal{COT}$ rings that include strictly more than three (resp., two) nodes. For the completeness of our work, we provided two algorithms: one using two robots exploring three nodes, and one for three or more robots moving among at least four nodes. These two algorithms allow to show that the necessary number of robots is also sufficient to solve the problem.

The cover time of any exploration algorithm in $\mathcal{COT}$ graphs is unbounded. However, our algorithm using three or more robots is speculative: even if its cover time is unbounded in $\mathcal{COT}$ rings, it is bounded in $\mathcal{ST}$ rings. More precisely, its cover time is asymptotically optimal in $\mathcal{ST}$ rings. Therefore, our algorithm solves the perpetual exploration problem in highly dynamic environments and it solves it in a very efficient way when being executed in static environments.

Our study is the first one dealing with the exploration problem in $\mathcal{COT}$ rings. Moreover, it is the first one considering the speculative aspect of algorithms in robot networks.

Chapter 10

Perpetual Exploration With Transient Faults

Contents

The exploration problem has never been studied while considering robots subject to transient faults (arbitrary faults such that there exists a time from which these faults no longer occur). In this chapter, we address the perpetual exploration problem in $\mathcal{COT}$ rings using robots that may be subject to transient faults.

More precisely, in this chapter, we characterize the number of robots (possibly subject to transient faults) necessary to solve the perpetual exploration problem in $\mathcal{COT}$ rings. Depending on these results, we describe some algorithms showing that the necessary number of self-stabilizing robots (i.e., of robots tolerating transient faults) is also sufficient to solve the perpetual exploration problem in $\mathcal{COT}$ rings. Such algorithms are called self-stabilizing algorithms [73].

Similarly as in the previous chapter, even if it is possible to solve the perpetual exploration problem in $\mathcal{COT}$ rings, the cover time of such algorithms is necessarily unbounded (refer to Section 8.3). This motivates the conception of a speculative algorithm to solve the perpetual exploration problem using robots that may be subject to transient faults. Indeed a speculative algorithm is an algorithm that must satisfy its requirements (i.e., the specification of the problem considered) in any execution in which it is susceptible to be executed but also that must be very efficient for the executions that are more likely to happen. In this chapter we consider only $\mathcal{ST}$ graphs (in addition to $\mathcal{COT}$ graphs) to prove the speculative aspect of our algorithm, i.e., we prove that the cover time of our algorithm is bounded (it has a cover time in $\Theta(n)$ rounds) and is asymptotically optimal when executed in $\mathcal{ST}$ graphs.

The results presented in this chapter have been published in SSS 2016 [27], and in Theoretical Computer Science [28].

Number of Robots	Size of Rings	Results
3	≥ 4	Possible (Theorem 10.3)
2	> 3	Impossible (Theorem 10.1)
	$= 3$	Possible (Theorem 10.5)
1	> 2	Impossible (Theorem 10.2)

Table 10.1: Overview of the results.

Results. The main contribution of this chapter is to prove that the necessary and sufficient numbers of robots for the perpetual exploration of highly dynamic rings exhibited in the previous chapter also hold in a self-stabilizing setting (see Table 10.1 for a summary of our results), at the price of the loss of anonymity of robots. Note that this price is unavoidable in the self-stabilizing setting since a classical symmetry breaking argument shows that it is impossible to solve the exploration problem with any number of anonymous robots. Indeed, a self-stabilizing algorithm is defined by Dijkstra [73] as an algorithm that recovers in a finite time a correct behavior regardless of the initial configuration of the system (which captures the effect of transient faults in the system). Hence, to be self-stabilizing, an exploration algorithm must tolerate any initial positions of robots. In particular, all robots may start on the same node. If the robots are anonymous, deterministic, and execute the same algorithm, no algorithm can prevent them to act as only one robot which is unable to perpetually explore a $\mathcal{COT}$ ring of size 3 and more (refer to Section 9.4).

Section 10.1 details the state of the art about self-stabilization. In Section 10.2, we present two impossibility results establishing that at least two (resp. three) self-stabilizing robots are necessary to perpetually explore $\mathcal{COT}$ rings of size greater than 3 (resp. 4) (even if robots are not anonymous). Note that these necessity results are not implied by the ones of the previous chapter (that focuses on anonymous robots). Then, Section 10.3 presents and proves the correctness of an algorithm solving the perpetual exploration problem in $\mathcal{COT}$ rings of size 4 and more in a self-stabilizing way with three robots. Section 10.4 proves the speculative aspect of this algorithm: even if its cover time in $\mathcal{COT}$ rings is unbounded, it is bounded (in $\Theta(n)$ rounds) and asymptotically optimal in $\mathcal{ST}$ rings. In Section 10.5, we present and prove the correctness of an algorithm showing the sufficiency of two robots to perpetually explore in a self-stabilizing manner $\mathcal{COT}$ rings of size three. Finally, Section 10.6 concludes the chapter.

In this chapter, we consider a system made of $\mathcal{R} = 3$ robots evolving in $\mathcal{COT}$ rings. In addition to the common assumptions made on robots all along the chapters of this thesis (see Section 3.2), we assume that the robots have not the same chirality. The robots have no prior knowledge about the graph in which they evolve nor on the robots. They are unable to directly communicate with each other by any means. They are endowed with strong local multiplicity detection meaning that they are able to detect the exact number of robots located on their current node. They are identified (each of them has a distinct identifier in a finite set ID). The robots know only their own identifier, they do not know the identifier of the other robots. They have persistent memory that is divided in two distinct areas: a corruptible one containing variables and an incorruptible one containing their algorithm and their constants (as their identifier). Call this model $\mathcal{M}_{self-stabilizing\ exploration}$ (also refer to Figure 3.5).

For the algorithms presented in this chapter, the variable dir of the robots can only take the values $right$ or $left$ (i.e., the variable dir cannot be equal to $\perp$).

10.1 Self-Stabilization

10.1.1 Definition

In the previous chapter, we consider that the robots always execute correctly their algorithm. However, it is well known that entities are not reliable: there exist periods in the execution when some entities do not execute correctly their algorithm, i.e., there exists faulty entities.

There are multiple kinds of faults classified according to three criteria [138]:

Span of the fault: a fault may be local (only one entity is faulty), partial (some of the entities but not all are faulty) or global (the whole system is impacted by the faults).

Nature of the fault: a fault induces an incorrect behavior which indicates its nature. Among the possible incorrect behavior, there is crash (the entity stops to perform its algorithm) [82], omission (the entity stops to send or to receive messages) [60], Byzantine (arbitrary fault) [119]...

Duration of the fault: a fault may be either permanent (there exists a time from which this fault always exists) or transient (there exists a time from which the fault does not occur anymore) or intermittent (the fault occurs infinitely often).

To specify a fault, each of the aforementioned criteria must be precised.

In this chapter, we consider that the robots may be subject to arbitrary (any span and any nature) transient faults. Hence, as indicated, there exists a time from which all the entities are never subject to fault anymore.

Once all the entities stop to be faulty, it is convenient if the system succeeds, without an external help, to find back a correct behavior. Dijkstra [73] introduced in 1974 self-stabilizing algorithms that recover in a finite time a correct behavior regardless of the initial configuration of the system (which captures the effect of transient faults in the system).

A self-stabilizing algorithm can be defined formally as follows (refer to Figure 10.1 for an illustration of an execution of a self-stabilizing algorithm).

Definition 10.1 (Self-stabilizing algorithm)**.** *An algorithm $\mathcal{A}$ is self-stabilizing for a problem $\mathcal{P}$ under a model $\mathcal{M}$ if and only if it ensures that, for any configuration γ_0, the execution of $\mathcal{A}$ under $\mathcal{M}$ starting from γ_0 contains a configuration γ_i such that the execution of $\mathcal{A}$ under $\mathcal{M}$ starting from γ_i satisfies the specification of $\mathcal{P}$.*

In other words, a self-stabilizing algorithm must guarantee two main properties:

Closure: there exists some configurations from which any execution of the algorithm satisfies the specification of the problem considered.

Convergence: starting from any configuration, any execution of the algorithm reaches, in finite time, a configuration that satisfies the closure property.

Since a self-stabilizing algorithm must be able to start its execution from any arbitrary configuration, this implies that a self-stabilizing algorithm using robots must tolerate both arbitrary initial states of the robots and arbitrary initial positions of the robots (in particular, robots may form towers in the initial configuration).

Note that being self-stabilizing imposes to tolerate arbitrary initialization of the variables, but it does not prevent the entities to possess constant values that cannot be corruptible by transient faults contrary to variables.

We give below the formal definition of a self-stabilizing algorithm using robots moving in evolving graphs.

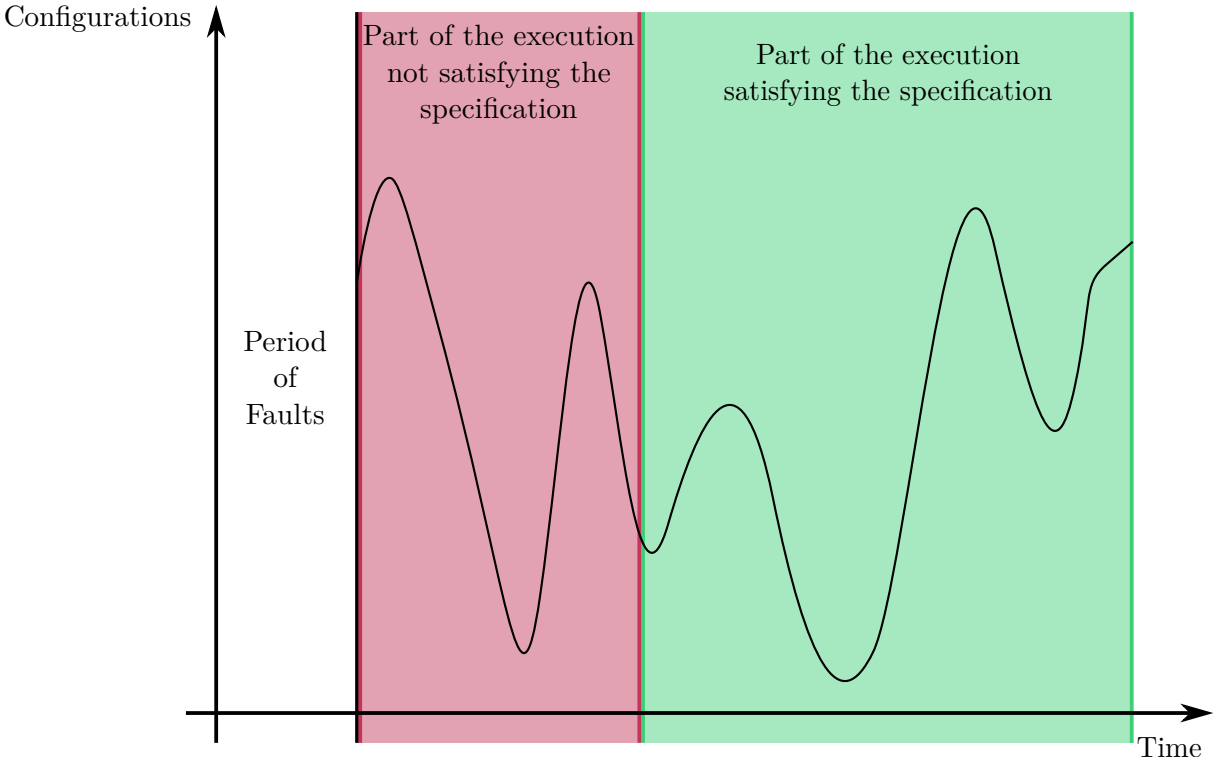


Figure 10.1: Example of an execution of a self-Stabilizing algorithm.

Definition 10.2 (Self-stabilizing algorithm using robots in evolving graphs)**.** *An algorithm $\mathcal{A}$ is self-stabilizing for a problem $\mathcal{P}$ under a model $\mathcal{M}$ in a class of evolving graphs $\mathcal{C}$ if and only if it ensures that, for any configuration γ_0, the execution of $\mathcal{A}$ under $\mathcal{M}$ in any evolving graph $\mathcal{G} \in \mathcal{C}$ starting from γ_0 contains a configuration γ_i such that the execution of $\mathcal{A}$ under $\mathcal{M}$ in $\mathcal{G}$ starting from γ_i satisfies the specification of $\mathcal{P}$.*

Even if self-stabilization permits to tolerate in an autonomous way very severe faults, during the time when the system is stabilizing there is no guarantee concerning the specification of the problem considered. Moreover, another drawback of this mechanism is that it is not possible for the robots to know when the system is stabilized: it is necessary to have a global view of the system to determine whether it is stabilized or not.

10.1.2 State of the Art

Although there exists a huge literature on self-stabilization in classical distributed systems, these work fall out of the scope of this thesis and we refer the interested reader to some reviews [73, 76, 138]. While considering robots, there are few articles focusing on self-stabilizing algorithms [70, 71, 129, 44]. Dieudonné and Petit [70] give a self-stabilizing probabilistic algorithm for scattering robots in the plane. Clement et al. [44] provide self-stabilizing probabilistic algorithms for the gathering and scattering problems using robots moving in the plane. Ooshita and Tixeuil [129] present probabilistic self-stabilizing algorithms for the gathering and a particular formation pattern problems using robots moving in undirected anonymous rings. Dieudonné and Petit [71] give a self-stabilizing deterministic algorithm to solve the gathering problem in the plane for an odd number of robots.

There exist multiple studies on the exploration problem in discrete static environment using oblivious robots [83, 84, 42, 64, 21]. These articles deal then with the arbitrary initial states of the robots since the robots do not store variables. However, none of them are self-stabilizing since

they impose special initial positions to the robots: initially there is at most one robot in each node of the graph. Moreover, Ooshita and Tixeuil [129] indicate that it is impossible to conceive a self-stabilizing (either deterministic or probabilistic) algorithm for the exploration with termination problem when robots are uniform, oblivious, anonymous, and not able to communicate directly.

There also exist some articles on exploration in discrete static environment using robots with memory [74, 97, 98]. However none of them is self-stabilizing: they do not tolerate arbitrary initialization of the variables.

When the methodology to explore a discrete dynamic environment is to use a random walk, like this is done by Avin et al. [11], then the algorithms are self-stabilizing. Indeed, the robots compute randomly at each instant time which is the next node to visit; hence the robots do not store any variable and they can be initially at any position in the graph. Therefore, there exist self-stabilizing probabilistic algorithms solving the exploration problem for robots evolving in discrete dynamic environment. Hence, this is also the case for robots evolving in discrete static environment.

Finally, among the articles dealing with deterministic exploration in discrete dynamic environment [89, 105, 106, 104, 125] (refer to Section 8.2 for more details about these articles) all consider robots endowed with memory and none of them are self-stabilizing.

To sum up, there is no self-stabilizing deterministic algorithm in the state of the art solving the exploration problem for robots evolving in discrete either static or dynamic environment.

10.2 Necessary Number of Robots

This section is devoted to the proof of the necessity of two (resp. three) self-stabilizing identified robots to perform perpetual exploration of highly dynamic rings of size at least 3 (resp. 4). To reach this goal, we provide two impossibility results.

First, we prove (see Theorem 10.1) that two robots with distinct identifiers are not able to perpetually explore in a self-stabilizing way $\mathcal{COT}$ rings of size greater than 4. Then, we show that we can borrow arguments from the previous chapter (see also [29]) to prove Theorem 10.2 that states that only one robot cannot complete the self-stabilizing perpetual exploration of $\mathcal{COT}$ rings of size greater than 3.

10.2.1 Highly Dynamic Rings of Size 4 or More

The proof of Theorem 10.1 is based on the construction of an adequate sequence of evolving graphs and the application of the generic framework proposed in [34] (see Section 4.2 for more information about this framework).

In order to build the evolving graphs sequence suitable for the proof of our impossibility result, we need the following technical lemma.

Lemma 10.1. *Any self-stabilizing deterministic perpetual exploration algorithm in $\mathcal{COT}$ rings of size 4 or more using 2 robots r_1 and r_2 with distinct identifiers satisfies: for any states s_1 and s_2, for any distinct identifiers id_{r_1} and id_{r_2}, it exists an integer t' ($t' \geq 0$) such that if r_1 (resp. r_2) with identifier id_{r_1} (resp. id_{r_2}) is on node u_1 (resp. u_2) in state s_1 (resp. s_2) at a time t and there exists only one adjacent edge to each position of the robots continuously present from time t to time $t + t'$, then r_1 and/or r_2 moves at time $t + t'$. This lemma holds even if the robots have the same chirality.*

Proof. By contradiction, assume that there exists a self-stabilizing deterministic perpetual exploration algorithm $\mathcal{A}$ in $\mathcal{COT}$ rings of size 4 or more using 2 robots r_1 and r_2 satisfying: there exist two states s_1 and s_2, and two distinct identifiers id_{r_1} and id_{r_2} such that, for any integer t' ($t' \geq 0$), r_1 (resp. r_2) with identifier id_{r_1} (resp. id_{r_2}) is on node u_1 (resp. u_2) in state s_1 (resp. s_2) at a time t and there exists only one adjacent edge to each position of the robots continuously present from time t to time $t + t'$ and none of the robots move at time $t + t'$.

Note that the existence of such states and identifiers is not contradictory with the definition of $\mathcal{A}$ if this algorithm has the ability to avoid that robots ever reach such states in each of its execution. In the following, we are going to prove that it is not the case.

Let G be a ring of size 4 or more. Let $\mathcal{G} = \{G_0, G_1, \ldots\}$ be a $\mathcal{COT}$ ring such that $\forall i, G_i = G$. Consider the evolving graph $\mathcal{G}'$ such that $\mathcal{G}' = \mathcal{G}\backslash\{(\{e\}, \{0, \ldots, +\infty\})\}$, where $e = \{u, v\}$ is an arbitrary edge of G. Note that $\mathcal{G}'$ is a $\mathcal{COT}$ ring, since it has only one eventual missing edge.

If $u_1 = u_2$, then we define the following configuration γ in $\mathcal{G}'$: r_1 (resp. r_2) with identifier id_{r_1} (resp. id_{r_2}) is on node u (resp. u) in state s_1 (resp. s_2).

If $u_1 \neq u_2$, then we define the following configuration γ in $\mathcal{G}'$: r_1 (resp. r_2) with identifier id_{r_1} (resp. id_{r_2}) is on node u (resp. v) in state s_1 (resp. s_2).

As $\mathcal{A}$ is self-stabilizing, there exists an execution ε of $\mathcal{A}$ in $\mathcal{G}'$ starting from γ. In both cases above, by construction, there is only one adjacent edge to each position of the robots continuously present from time 0 to $+\infty$, and r_1 and r_2 are respectively in state s_1 and s_2 at time 0. Then, by assumption, r_1 and r_2 do not leave their respective nodes in ε. As $\mathcal{G}'$ counts 4 nodes or more, we obtain a contradiction with the fact that $\mathcal{A}$ is a self-stabilizing algorithm solving deterministically the perpetual exploration problem for $\mathcal{COT}$ rings of size 4 or more using two robots possessing distinct identifiers. □

Theorem 10.1. *There exists no deterministic self-stabilizing algorithm satisfying the perpetual exploration problem under $\mathcal{M}_{self-stabilizing\ exploration}$ in $\mathcal{COT}$ rings of size 4 or more with two robots.*

Proof. By contradiction, assume that there exists a deterministic algorithm $\mathcal{A}$ satisfying the perpetual exploration specification in a self-stabilizing way in any $\mathcal{COT}$ ring of size 4 or more using two robots r_1 and r_2 possessing distinct identifiers.

Consider the $\mathcal{COT}$ graph $\mathcal{G} = \{G_0, G_1, \ldots\}$ whose footprint G is a ring of size 4 or more and such that $\forall i \in \mathbb{N}, G_i = G$.

Consider four nodes u, v, w and x of $\mathcal{G}$, such that node v is the adjacent node of u in the clockwise direction, w is the adjacent node of v in the clockwise direction, and x is the adjacent node of w in the clockwise direction. We denote respectively e_{vr} and e_{vl} the clockwise and counter-clockwise adjacent edges of v, e_{wr} and e_{wl} the clockwise and counter-clockwise adjacent edges of w, and e_{xr} and e_{xl} the clockwise and counter-clockwise adjacent edges of x. Note that $e_{vr} = e_{wl}$, and $e_{wr} = e_{xl}$.

Let $\mathcal{G}'$ be $\mathcal{G}\backslash\{(\{e_{vl}\}, \mathbb{N})\}$. Let ε be the execution of $\mathcal{A}$ in $\mathcal{G}'$ starting from the configuration where r_1 (resp. r_2) is located on node v (resp. w).

Our goal is to construct a sequence of $\mathcal{COT}$ rings denoted $(\mathcal{G}_m)_{m\in\mathbb{N}}$ such that $\mathcal{G}_0 = \mathcal{G}'$ and, for any $i \geq 0$, only nodes among $\{v, w, x\}$ have been visited until time t_i in ε_i (the execution of $\mathcal{A}$ in $\mathcal{G}_i$ starting from the same configuration as ε), $(t_m)_{m\in\mathbb{N}}$ being a strictly increasing sequence with $t_0 = 0$. First, we show in the next paragraph that, if some such $\mathcal{G}_i$ exists and, moreover, ensures the existence of an integer $t'_i \geq 0$ such that until time $t_i + t'_i + 1$ only nodes among $\{v, w, x\}$ have been visited in ε_i, then we can construct $\mathcal{G}_{i+1}$ as shown in Figure 10.2. We prove, after that, that our construction guarantees the existence of such a t'_i, implying the well definition of $(\mathcal{G}_m)_{m\in\mathbb{N}}$.

Since $\mathcal{G}_i$ is a $\mathcal{COT}$ ring, and since $\mathcal{A}$ is a deterministic algorithm solving the perpetual exploration problem in a self-stabilizing way in $\mathcal{COT}$ rings of size 4 or more using 2 robots possessing distinct identifiers, when the configuration γ_i at time t_i is such that there is exactly one adjacent edge present to the location of each of the two robots, we use Lemma 10.1 to exhibit the smallest integer $t'_i \geq 0$ such that if the configuration γ_i lasts from time t_i to time $t_i + t'_i$, then one or both of the robots move at time $t_i + t'_i$. $(*)$ Similarly, when the configuration γ_i at time t_i is such that there is only one missing edge, and that only one robot is adjacent to this missing edge, then we can also exhibit the smallest integer $t'_i \geq 0$ such that at time $t_i + t'_i$ at least one of the robots moves. Indeed, if this configuration lasts from time t_i to time $+\infty$, $\mathcal{G}_i$ is a $\mathcal{COT}$ ring, and if none of the robots move in this configuration, the perpetual exploration cannot be solved. Therefore

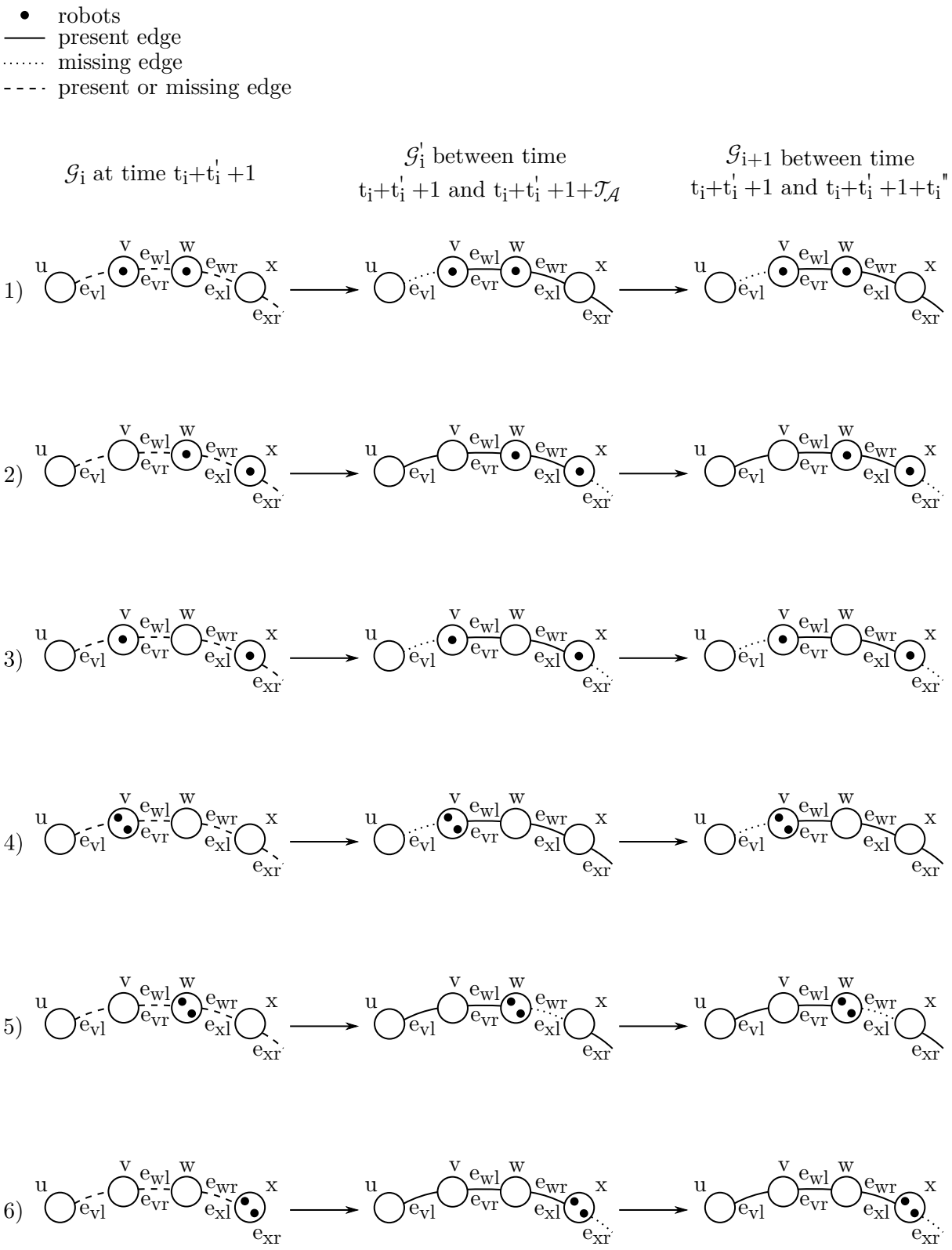


Figure 10.2: Construction of $\mathcal{G}_{i+1}$ from $\mathcal{G}_i$.

such an integer t'_i exists. Let $\mathcal{T}_\mathcal{A}$ be the largest integer t'_i whatever the states and identifiers considered by the robots executing $\mathcal{A}$. Since the states reached by the robots executing $\mathcal{A}$ and the identifiers are among a finite set of values, $\mathcal{T}_\mathcal{A}$ exists and is bounded.

In the following we show how we construct the dynamic graph $\mathcal{G}_{i+1}$ in function of $\mathcal{G}_i$, t_i and t'_i. As we assume that in $\mathcal{G}_i$, until time $t_i + t'_i + 1$, only nodes among $\{v, w, x\}$ have been visited, then the following cases are possible. If the two robots are on two distinct nodes in $\mathcal{G}_i$ at time $t_i + t'_i + 1$ then:

1. if, at time $t_i + t'_i + 1$ in $\mathcal{G}_i$, one of the robots is on node v and the other robot is on node w then let $E_i = \{e_{vl}\}$.
2. if, at time $t_i + t'_i + 1$ in $\mathcal{G}_i$, one of the robots is on node w and the other robot is on node x then let $E_i = \{e_{xr}\}$.
3. if, at time $t_i + t'_i + 1$ in $\mathcal{G}_i$, one of the robots is on node v and the other robot is on node x then let $E_i = \{e_{vl}, e_{xr}\}$.

If the two robots are on the same node in $\mathcal{G}_i$ at time $t_i + t'_i + 1$ then:

4. if, at time $t_i + t'_i + 1$ in $\mathcal{G}_i$, the two robots are on node v, then let $E_i = \{e_{vl}\}$.
5. if, at time $t_i + t'_i + 1$ in $\mathcal{G}_i$, the two robots are on node w then let $E_i = \{e_{wr}\}$.
6. if, at time $t_i + t'_i + 1$ in $\mathcal{G}_i$, the two robots are on node x, then let $E_i = \{e_{xr}\}$.

For each case, we define $\mathcal{G}'_i$ such that $\mathcal{G}'_i$ and $\mathcal{G}_i$ have the same footprint and $\mathcal{G}'_i = \mathcal{G}_i \otimes_{t_i+t'_i} (\mathcal{G}\backslash\{(E_i, \{t_i+t'_i+1, \ldots, t_i+t'_i+1+\mathcal{T}_\mathcal{A}\})\})$. Note that $\mathcal{G}'_i$ is a $\mathcal{COT}$ ring since it is indistinguishable from $\mathcal{G}$ after $t_i + t'_i + 1 + \mathcal{T}_\mathcal{A}$. We can then apply either Lemma 10.1 or $(*)$ to find the smallest integer t_i" (with $\mathcal{T}_\mathcal{A} \geq t_i" \geq 0$) such that at time $t_i + t'_i + 1 + t_i$" at least one of the robots moves in $\mathcal{G}'_i$. By definition of $\mathcal{T}_\mathcal{A}$, t_i" exists. Finally, we define $\mathcal{G}_{i+1}$ such that $\mathcal{G}_{i+1}$ and $\mathcal{G}_i$ have the same footprint and $\mathcal{G}_{i+1} = \mathcal{G}_i \otimes_{t_i+t'_i} (\mathcal{G}\backslash\{(E_i, \{t_i + t'_i + 1, \ldots, t_i + t'_i + 1 + t_i"\})\})$.

$\mathcal{G}_{i+1}$ is a $\mathcal{COT}$ ring (since it is indistinguishable from $\mathcal{G}$ after $t_i + t'_i + 1 + t_i$"). Note that $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ are indistinguishable for robots until time $t_i + t'_i$. This implies that, at time $t_i + t'_i$, r_1 and r_2 are on the same node in ε_i and in ε_{i+1}. By construction of t'_i, either r_1 or r_2 or both of the two robots move at time $t_i + t'_i$ in ε_{i+1}. Moreover, even if one or both of the robots move during the Move phase of time $t_i + t'_i$, at time $t_i + t'_i + 1$ the robots are still on nodes among $\{v, w, x\}$, by assumption on $\mathcal{G}_i$ and construction of $\mathcal{G}_{i+1}$, since from time t_i to time $t_i + t'_i$ the edges permitting a robot to go on a node other than the nodes among $\{v, w, x\}$ are missing.

Let $t_{i+1} = t_i + t'_i + 1$ and $t_i" = t'_{i+1}$. Then we can construct recursively each dynamic ring of $(\mathcal{G}_m)_{m\in\mathbb{N}}$ by applying the argues above on all the possible configurations reached by the movements of the robots at time $t_{i+1} + t'_{i+1} + 1$ in $\mathcal{G}_{i+1}$.

Note that the recurrence can be initiated, since $\mathcal{G}_0$ exists, is a $\mathcal{COT}$ ring and from time $t_0 = 0$ to time $t_0 + t'_0 + 1$ only nodes among $\{v, w, x\}$ have been visited. In other words, $(\mathcal{G}_m)_{m\in\mathbb{N}}$ is well-defined.

We can then define the evolving graph $\mathcal{G}_\omega$ such that $\mathcal{G}_\omega$ and $\mathcal{G}_0$ have the same footprint and for all $i \in \mathbb{N}$, $\mathcal{G}_\omega$ shares a common prefix with $\mathcal{G}_i$ until time $t_i + t'_i$. As the sequence $(t_m + t'_m)_{m\in\mathbb{N}}$ is increasing by construction, this implies that the sequence $(\mathcal{G}_m)_{m\in\mathbb{N}}$ converges to $\mathcal{G}_\omega$.

Note that $\mathcal{G}_\omega$ is a $\mathcal{COT}$ ring. Indeed, among the cases presented, only the dynamic graph $\mathcal{G}_{i+1}$ built in case 3 possesses two simultaneous missing edges between time t_{i+1} and $t_{i+1} + t'_{i+1}$. However, even if at some point such a dynamic ring is used to construct $(\mathcal{G}_m)_{m\in\mathbb{N}}$, note that since at least one of the two robots moves at time $t_{i+1} + t'_{i+1}$ then $\mathcal{G}_{i+2}$ cannot be built thanks to case 3. Therefore, there is only one missing edge in $\mathcal{G}_{i+2}$ between time t_{i+2} and $t_{i+2} + t'_{i+2}$. In conclusion, there is at most one eventual missing edge and $\mathcal{G}_\omega$ is a $\mathcal{COT}$ ring.

Applying the theorem of [34], we obtain that, until time $t_i + t'_i$, the execution of $\mathcal{A}$ in $\mathcal{G}_\omega$ is identical to the one in $\mathcal{G}_i$. This implies that, executing $\mathcal{A}$ in $\mathcal{G}_\omega$ (whose footprint is a ring of size 4

or more), r_1 and r_2 only visit the nodes among $\{v, w, x\}$. This is contradictory with the fact that $\mathcal{A}$ satisfies the perpetual exploration specification in $\mathcal{COT}$ rings of size 4 or more using 2 robots possessing distinct identifiers. □

10.2.2 Highly Dynamic Rings of Size 3 or More

In the previous chapter, we prove (in Theorem 10.2) that a single anonymous and synchronous robot cannot perpetually explore $\mathcal{COT}$ rings of size 3 or more in a fault-free setting. We can do two observations. First, any fault-free synchronous execution is possible in a self-stabilizing setting. Second, in the case of a *single* robot, the anonymous and the identified model are equivalent.

These observations are sufficient to directly state the following result:

Theorem 10.2. *There exists no deterministic self-stabilizing algorithm satisfying the perpetual exploration problem under $\mathcal{M}_{self-stabilizing\ exploration}$ in $\mathcal{COT}$ rings of size 3 or more using one robot.*

10.3 Sufficiency of Three Robots for $n \geq 4$

In this section, we present our self-stabilizing deterministic algorithm for the perpetual exploration of any $\mathcal{COT}$ ring of size greater than four with three robots. In this context, the difficulty to complete the exploration is twofold. First, in $\mathcal{COT}$ graphs, robots must deal with the possible existence of some eventual missing edge (without the guarantee that such edge always exists). Note that, in the case of a ring, there is at most one eventual missing edge in any execution (otherwise, we have a contradiction with the connected-over-time property). Second, robots have to handle the arbitrary initialization of the system (corruption of variables and arbitrary position of robots).

10.3.1 Presentation of the algorithm

Principle of the algorithm. The main idea behind our algorithm is that a robot does not change its direction (arbitrarily initialized) while it is isolated. This allows robots to perpetually explore $\mathcal{COT}$ rings with no eventual missing edge regardless of the initial direction of the robots.

Obviously, this idea is no longer sufficient when there exists an eventual missing edge since, in this case, at least two robots will eventually be stuck (i.e., they point to an eventual missing edge that they are never able to cross) forever at one end of the eventual missing edge. Indeed, there are three robots in the system, hence at least two of them consider initially the same global direction, and hence will be stuck on the same extremity of the eventual missing edge. When two (or more) robots are located at the same node, we say that they form a tower. In this case, our algorithm ensures that at least one robot leaves the tower in a finite time using its identifier (see below). In this way, we obtain that, in a finite time, a robot is stuck at each end of the eventual missing edge. These two robots located at two ends of the eventual missing edge play the role of "sentinels" while the third one (we call it a "visitor") visits other nodes of the ring in the following way. The "visitor" keeps its direction until it meets one of these "sentinels," they then switch their roles: After the meeting, the "visitor" still maintains the same direction (becoming thus a "sentinel") while the "sentinel" robot changes its direction (becoming thus a "visitor" until reaching the other "sentinel").

In fact, robots are never aware if they are actually stuck at an eventual missing edge or are just temporarily stuck on an edge that will reappear in a finite time. Hence, robots are never aware of their status (sentinel or visitor). That is why it is important that the robots keep their directions and try to move forward while there is no meeting in order to track a possible eventual missing edge. Our algorithm only guarantees convergence in a finite time towards a configuration where a robot plays the role of "sentinel" at each end of the eventual missing edge if such an edge

exists. Note that, in the case where there is no eventual missing edge, this mechanism does not prevent the correct exploration of the ring since it is impossible for a robot to be stuck forever.

Our algorithm easily deals with the initial corruption of its variables. Indeed, all variables of a robot (with the exception of a counter and the variable *dir* whose initial respective values have no particular impact) store information about the environment of this robot in the previous round it was edge-activated (i.e., in the previous round where it was on a node that possesses a present adjacent edge, refer to Section 3.2). These variables are updated each time a robot is edge-activated. Since we consider $\mathcal{COT}$ rings, there can only exist one eventual missing edge, therefore all robots are infinitely often edge-activated. The initial values of these variables are hence reset in a finite time. The main difficulty to achieve self-stabilization is to deal with the arbitrary initial position of robots. In particular, the robots may initially form towers. In the worst case, all robots of a tower may be stuck at an eventual missing edge and be in the same state. They are then unable to start the "sentinels"/"visitor" scheme explained above. Our algorithm needs to "break" such a tower in a finite time (i.e., one robot must leave the node where the tower is located). In other words, we tackle a classical problem of symmetry breaking. We achieve this by providing each robot with a function that returns, in a finite number of invocations, different global directions to two robots of the tower based on the private identifier of the robot and without any communication among the robots. More precisely, this is done thanks to a transformation of the robot identifier: each bit of the binary representation of the identifier is duplicated and we add the bits "010" at the end of the sequence of these duplicated bits. Then, at each invocation of the function, a robot reads the next bit of this transformed identifier. If the robot reads zero, it tries to move to its left. Otherwise, it tries to move to its right. Doing so, in a finite number of invocation of this function, at least one robot leaves the tower due to the construction of the transformed identifiers that roughly guarantees some non-periodicity properties. If necessary, we repeat this "tower breaking" scheme until we are able to start the "sentinels"/"visitor" scheme.

The main difficulty in designing this algorithm is to ensure that these two mechanisms ("sentinels"/"visitor" and "tower breaking") do not interfere with each other and prevent the correct exploration. We solve this problem by adding some waiting "at good time," especially before starting the procedure of tower breaking by identifier to ensure that robots do not prematurely turn back and "forget" to explore some parts of the ring.

Formal presentation of the algorithm. Before presenting our algorithm formally, we need to introduce the set of constants (i.e., variables assumed not to be corruptible) and the set of variables of each robot. We also introduce three auxiliary functions.

As stated in the model, each robot r has a unique identifier. We denote it by id_r and represent it in binary as $b_1 b_2 \dots b_{|id_r|}$. We define, for the purpose of the "breaking tower" scheme, the constant $TransformedIdentifier$ by its binary representation $b_1 b_1 b_2 b_2 \dots b_{|id_r|} b_{|id_r|} 010$ (each bit of id_r is duplicated and we add the three bits 010 at the end). We store the length of the binary representation of $TransformedIdentifier$ in the constant ℓ and we denote its i^{th} bit by $TransformedIdentifier[i]$ for any $1 \leq i \leq \ell$.

In addition to the variable *dir* defined in the model, each robot has the following three variables: (i) the variable $i \in \mathbb{N}$ corresponds to an index to store the position of the last bit read from $TransformedIdentifier$; (ii) the variable $NumberRobotsPreviousEdgeActivation \in \mathbb{N}$ stores the number of robots that were present at the node of the robot during the Look phase of the last round where it was edge-activated; and (iii) the variable $HasMovedPreviousEdgeActivation \in \{true, false\}$ indicates if the robot has crossed an edge during its last edge-activation.

Our algorithm makes use of a function Update() that updates the values of the two last variables according to the current environment of the robot each time it is edge-activated. We provide the pseudo-code of this function in Algorithm 10.1. Note that this function also allows us to deal with the initial corruption of the two last variables since it resets them in the first round where the robot is edge-activated.

We already stated that, whenever robots are stuck forming a tower, they make use of a function to "break" the tower in a finite time. The pseudo-code of this function GIVEDIRECTION() appears in Algorithm 10.2. It assigns the value $left$ or $right$ to the variable dir of the robot depending on the i^{th} bit of the value of $TransformedIdentifier$. The variable i is incremented modulo ℓ (that implicitly resets this variable when it is corrupted) to ensure that successive calls to GIVEDIRECTION() will consider each bit of $TransformedIdentifier$ in a round-robin way. As shown in the next section, this function guarantees that, if two robots are stuck together in a tower and invoke repeatedly their own function GIVEDIRECTION(), then two distinct global directions are given in finite time to the two robots regardless of their chirality. This property allows the algorithm to "break" the tower since at least one robot is then able to leave the node where the tower is located.

Algorithm 10.1 Function UPDATE()

```
1: function UPDATE()
2:     if ExistsAdjacentEdge() then
3:         NumberRobotsPreviousEdgeActivation := NumberOfRobotsOnNode()
4:         HasMovedPreviousEdgeActivation := ExistsEdge(dir, current)
5:     end if
6: end function
```

Finally, we define the function OPPOSITEDIRECTION() that simply affects the value $left$ (resp. $right$) to the variable dir when $dir = right$ (resp. $dir = left$).

There are two types of configurations in which the robots may change the direction they consider. So, our algorithm needs to identify them. We do so by defining a predicate that characterizes each of these configurations.

The first one, called $WeAreStuckInTheSameDirection()$, is dedicated to the detection of configurations in which the robot must invoke the "tower breaking" mechanism. Namely, the robot is stuck since at least one edge-activation with at least another robot and the edge in the direction opposite to the one considered by the robot is present. More formally, this predicate is defined as follows:

$$\begin{aligned} &WeAreStuckInTheSameDirection() \equiv \\ &\quad (NumberOfRobotsOnNode() > 1) \\ &\wedge\ (NumberOfRobotsOnNode() = NumberRobotsPreviousEdgeActivation) \\ &\wedge\ \neg ExistsEdge(dir, current) \\ &\wedge\ ExistsEdge(\overline{dir}, current) \\ &\wedge\ \neg HasMovedPreviousEdgeActivation \end{aligned}$$

The second predicate, called $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$, is designed to detect configurations in which the robot must transition from the "sentinel" to the "visitor" role in the "sentinel"/"visitor" scheme. More precisely, such configuration is characterized by the fact that the robot is edge-activated, stuck during its previous edge-activation, and there are strictly more robots located at its node than at its previous edge-activation. More formally, this predicate is defined as follows:

$$\begin{aligned} &IWasStuckOnMyNodeAndNowWeAreMoreRobots() \equiv \\ &\quad (NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation) \\ &\wedge\ \neg HasMovedPreviousEdgeActivation \\ &\wedge\ ExistsAdjacentEdge() \end{aligned}$$

Now, we are ready to present the pseudo-code of the core of our algorithm (called $\mathbb{SS}\_\mathbb{PEF}\_3$, see Algorithm 10.3). The basic idea of the algorithm is the following. The function GIVEDIRECTION() is invoked when $WeAreStuckInTheSameDirection()$ is true (to try to "break" the tower after the appropriate waiting), while the function OPPOSITEDIRECTION() is called when $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ is true (to implement the "sentinel"/"visitor"

Algorithm 10.2 Function GiveDirection()

```
1: function GiveDirection()
2:     i := i + 1 (mod ℓ) + 1
3:     if TransformedIdentifier[i] = 0 then
4:         dir := left
5:     else
6:         dir := right
7:     end if
8: end function
```

scheme). Afterwards, the function Update() is called (to update the state of the robot according to its environment).

Algorithm 10.3 $\mathbb{SS}\_\mathbb{PEF}\_3$

```
1: StuckTogether :: WeAreStuckInTheSameDirection()
2:         ⟶ GiveDirection() ; Update()
3: StuckAndJoined :: IWasStuckOnMyNodeAndNowWeAreMoreRobots()
4:         ⟶ OppositeDirection() ; Update()
5: Update :: true ⟶ Update()
```

10.3.2 Preliminaries to the Correctness Proof

First, we introduce some preliminary results that are used in the proof. All along the proof of correctness, we extensively use the notions of short-lived tower (robots located on a same node that disagree on their global direction the first time there exists an adjacent edge to their current location) and long-lived tower (robots located on a same node that agree on their global direction at each instant time (before the tower breaks) there exists an adjacent edge to their current location and that do not form a short-lived tower) that we defined in Section 3.2.4.

As there are only three robots in our system, and that in each round each of them considers a global direction, we can make the following observation.

Observation 10.1. *There are at least two robots having the same global direction at each instant time.*

In the remainder of this section, we consider an execution $\mathcal{E}$ of $\mathbb{SS}\_\mathbb{PEF}\_3$ executed by three robots r_1, r_2, and r_3 in a $\mathcal{COT}$ ring $\mathcal{G}$ of size $n \in \mathbb{N}$, with $n \geq 4$, starting from an arbitrary configuration.

We say that a robot r has a coherent state at time t, if the value of its variable $NumberRobots\text{-}PreviousEdgeActivation_{r,t}$ (i.e., value of the variable $NumberRobotsPreviousEdgeActivation$ of the robot r after the Look phase of round t, refer to Section 3.2.1 for the introduction of this notation) corresponds to the value of its predicate $NumberOfRobotsOnNode()$ at its previous edge-activation and the value of its variable $HasMovedPreviousEdgeActivation_{r,t}$ corresponds to the value of its predicate $ExistsEdge(dir, current)$ at its previous edge-activation. The following lemma states that, for each robot, there exists a suffix of the execution in which the state of the robot is coherent.

Lemma 10.2. *For any robot, there exists a time from which its state is always coherent.*

Proof. Consider a robot r performing $\mathbb{SS}\_\mathbb{PEF}\_3$.

$\mathcal{G}$ is a $\mathcal{COT}$ ring, hence at least one adjacent edge to each node of $\mathcal{G}$ is infinitely often present. This implies that r is infinitely often edge-activated, whatever its location is. Let t be the first time at which r is edge-activated.

Variables can be updated only during Compute phases of rounds. When executing $\mathbb{SS}\_\mathbb{PEF}\_3$, the variables $NumberRobotsPreviousEdgeActivation$ and $HasMovedPreviousEdgeActivation$

of r are updated with the current values of its predicates $NumberOfRobotsOnNode()$ and $ExistsEdge(dir, current)$ only when it is edge-activated.

Therefore from time $t+1$, r is in a coherent state. □

Let t_1, t_2, and t_3 be respectively the time at which the robots r_1, r_2, and r_3, respectively are in a coherent state. Let $t_{max} = max\{t_1, t_2, t_3\}$. From Lemma 10.2, the three robots are in a coherent state from t_{max}. In the remainder of the proof, we focus on the suffix of the execution after t_{max}.

The two following lemmas (in combination with Lemma 4.5 and Corollary 4.1) aim at showing that, regardless of the chirality of the robots and the initial values of their variables i, a finite number of synchronous invocations of the function GiveDirection() by two robots of a long-lived tower returns them a distinct global direction. This property is shown by looking closely to the structure of the binary representation of the transformed identifiers of the robots.

Indeed, the round-robin reading of its transformed identifier by a robot r can be seen as an infinite binary word $U(r)$ (the infinite concatenation of this transformed identifier). When robots of a long-lived tower are stuck on a node, we want them to break the tower by considering simultaneously two opposite global directions in finite time. The robots of a long-lived tower that are stuck on a node have their predicate $WeAreStuckInTheSameDirection()$ true and hence use the function GiveDirection(). Two robots that possess the same chirality and that call the function GiveDirection() must consider two distinct bits at the same time to consider two opposite global directions at this time. Lemma 10.3 shows that any common factor of $U(r_1)$ and $U(r_2)$ is finite for any pair of robots r_1 and r_2, allowing us to later (see Corollary 10.1) state that, in finite time, if the predicate $WeAreStuckInTheSameDirection()$ of the robots are infinitely often true then the long-lived tower they are involved in is broken. Lemma 10.4 shows a similar result when robots of the long-lived tower does not have the same chirality.

To state formally these lemmas, we need to introduce some vocabulary and definitions from combinatorics on words. We consider words as (possibly infinite) sequence of letters from the alphabet $A = \{0, 1\}$. Given a word u, we refer to its i^{th} letter by $u[i]$. The length of a word u (denoted $|u|$) is its number of letters. Given two finite words $u = u[1] \dots u[k]$ and $v = v[1] \dots v[\ell]$ (with $k = |u|$ and $\ell = |v|$), the concatenation of u and v (denoted $u.v$) is the word $u[1] \dots u[k]v[1] \dots v[\ell]$ (with $|u.v| = k + \ell$). Given a finite word u, the word u^1 is u itself and the word u^z ($z > 1$) is the word $u.u^{z-1}$. Given a finite word u, the word u^ω is the infinite word $u.u.u.\dots$. A prefix u_1 of a word u is a word such that there exists a word u_2 satisfying $u = u_1.u_2$. A suffix u_2 of a word u is a word such that there exists a word u_1 satisfying $u = u_1.u_2$. A factor u_2 of a word u is a word such that there exists a prefix u_1 and a suffix u_3 of u satisfying $u = u_1.u_2.u_3$. The factor of u starting from the i^{th} bit of u and ending to the j^{th} bit of u included is denoted $u[i \dots j]$. A circular permutation of a word u is a word of the form $u_2.u_1$ where $u = u_1.u_2$. Let us introduce the notation $\overline{w}$ which given a word w is defined such that $\overline{w} = \prod_{i \in \{1,\dots,|w|\}} \overline{w[i]}$ where if $w[i] = 1$ then $\overline{w[i]} = 0$, and if $w[i] = 0$ then $\overline{w[i]} = 1$.

Lemma 10.3. *Let u and v be two distinct transformed identifiers. If u^ω and v^ω share a common factor X, then X is finite.*

Proof. Consider two transformed identifiers u and v such that $u \neq v$.

By definition, the transformed identifier u is either equal to 00.010 or to $11.\Pi_{d=1}^{\alpha(u)}(\Pi_1^{\beta(u,d)}00.\Pi_1^{\gamma(u,d)}11).010$ $(*)$ with $\alpha(u)$ a function giving the number of blocks $(\Pi_1^{\beta(u,d)}00.\Pi_1^{\gamma(u,d)}11)$ contained in u, $\beta(u,d)$ a function giving the number of pairs of bits 00 contained in the d^{th} block of u, and $\gamma(u,d)$ a function giving the number of pairs of bits 11 contained in the d^{th} block of u.

Similarly, by definition v is either equal to 00.010 or to $11.\Pi_{d=1}^{\alpha(v)}(\Pi_1^{\beta(v,d)}00.\Pi_1^{\gamma(v,d)}11).010$ $(**)$.

Let $U = u^\omega$ and $V = v^\omega$.

Assume by contradiction that U and V share a common factor X of infinite size. Hence $U = x.X$ and $V = y.X$, with x (resp. y) the prefix of U (resp. of V). We have $X = \tilde{u}^\omega$, where

$\tilde{u}$ is a circular permutation of the word u, and $X = \tilde{v}^\omega$, where $\tilde{v}$ is a circular permutation of the word v.

By definition of a common factor we have $\forall h \in \mathbb{N}^*, U[|x| + h] = V[|y| + h]$ $(***)$.

Let $k \in \mathbb{N}^*$ such that $U[|x| + k] = 0$, $U[|x| + k + 1] = 1$ and $U[|x| + k + 2] = 0$. By $(*)$ and since $U = x.X = x.\tilde{u}^\omega$, k exists. By $(*)$ and by construction of U, we know that $U[|x| + k + 3 \dots |x| + k + |u| + 2]$ is equal to u and $U[|x| + k + 3 \dots |x| + k + |u| - 1]$ is either equal to 00 or to $11.\Pi_{d=1}^{\alpha(u)}(\Pi_1^{\beta(u,d)}00.\Pi_1^{\gamma(u,d)}11)$.

By $(***)$, we have $V[|y| + k] = 0$, $V[|y| + k + 1] = 1$ and $V[|y| + k + 2] = 0$. By $(**)$ and by construction of V, we know that $V[|y| + k + 3 \dots |y| + k + |v| + 2]$ is equal to v and $V[|y| + k + 3 \dots |y| + k + |v| - 1]$ is either equal to 00 or to $11.\Pi_{d=1}^{\alpha(v)}(\Pi_1^{\beta(v,d)}00.\Pi_1^{\gamma(v,d)}11)$.

Case 1: $|u| = |v|$.

If $|u| = |v|$, then by $(***)$ we have $U[|x| + k + 3 \dots |x| + k + |u| + 2] = V[|y| + k + 3 \dots |y| + k + |v| + 2]$. This implies that $u = v$, which leads to a contradiction with the fact that u and v are distinct.

Case 2: $|u| \neq |v|$.

Without lost of generality assume that $|u| < |v|$. We have $U[|x| + k + |u|] = 0$, $U[|x| + k + |u| + 1] = 1$ and $U[|x| + k + |u| + 2] = 0$. Therefore by $(***)$ we have $V[|y| + k + |u|] = 0$, $V[|y| + k + |u| + 1] = 1$ and $V[|y| + k + |u| + 2] = 0$.

Note that $|u| = 2w + 3$ with $w \in \mathbb{N}^*$. Similarly $|v| = 2z + 3$, with $z \in \mathbb{N}^*$, and $z > w$ since $|u| < |v|$. Since $V[|y| + k + 3 \dots |y| + k + |v| + 2]$ is equal to v, this implies that $V[|y| + k + 3] = v[1]$, and $V[|y| + k + |u|] = V[|y| + k + 2w + 3] = v[i]$ where i is odd and such that $1 \leq i \leq 2z$. Hence by $(**)$, necessarily $V[|y| + k + |u|] = V[|y| + k + |u| + 1]$, which leads to a contradiction with the fact that $V[|y| + k + |u|] = 0$ and $V[|y| + k + |u| + 1] = 1$.

□

Lemma 10.4. *Let u and v be two distinct transformed identifiers. If u^ω and $\overline{v}^\omega$ share a common factor X, then X is finite.*

Proof. Consider two transformed identifiers u and v such that $u \neq v$.

By definition, the transformed identifier u is either equal to 00.010 or to $11.\Pi_{d=1}^{\alpha(u)}(\Pi_1^{\beta(u,d)}00.\Pi_1^{\gamma(u,d)}11).010$ $(*)$ with $\alpha(u)$ a function giving the number of blocks $(\Pi_1^{\beta(u,d)}00.\Pi_1^{\gamma(u,d)}11)$ contained in u, $\beta(u,d)$ a function giving the number of pairs of bits 00 contained in the d^{th} block of u, and $\gamma(u,d)$ a function giving the number of pairs of bits 11 contained in the d^{th} block of u.

Similarly, by definition v is either equal to 00.010 or to $11.\Pi_{d=1}^{\alpha(v)}(\Pi_1^{\beta(v,d)}00.\Pi_1^{\gamma(v,d)}11).010$. Call $w = \overline{v}$. This implies that $|w| = |v|$ and w is either equal to 11.101 or to $00.\Pi_{d=1}^{\alpha(v)}(\Pi_1^{\beta(v,d)}11.\Pi_1^{\gamma(v,d)}00).101$ $(**)$. Note that u and w are distinct. Indeed, if $|u| \neq |v|$ then, w and u are distinct since $|w| = |v|$. If $|u| = |v|$ then, since the suffix of size 3 of u is the word 010, and the suffix of size 3 of w is the word 101, then u and w are distinct.

Let $U = u^\omega$ and $W = w^\omega$.

Assume by contradiction that U and W share a common factor X of infinite size. Hence $U = x.X$ and $W = y.X$, with x (resp. y) the prefix of U (resp. of W). We have $X = \tilde{u}^\omega$, where $\tilde{u}$ is a circular permutation of the word u, and $X = \tilde{w}^\omega$, where $\tilde{w}$ is a circular permutation of the word w.

By definition of a common factor we have $\forall h \in \mathbb{N}^*, U[|x| + h] = W[|y| + h]$ $(***)$.

Let $k \in \mathbb{N}^*$ such that $U[|x| + k] = 0$, $U[|x| + k + 1] = 1$ and $U[|x| + k + 2] = 0$. By $(*)$ and since $U = x.X = x.\tilde{u}^\omega$, k exists. By $(*)$ and by construction of U, we know that $U[|x| + k + 3 \dots |x| + k + |u| + 2]$ is equal to u.

By $(***)$, we have $W[|y|+k]=0$, $W[|y|+k+1]=1$ and $W[|y|+k+2]=0$. By $(**)$ and by construction of W, we know that either $W[|y|+k+4\ldots|y|+k+|w|+3]=w$ (in the case where $W[|y|+k+1]=w[|w|-2]$) or $W[|y|+k+2\ldots|y|+k+|w|+1]=w$ (in the case where $W[|y|+k+1]=w[|w|]$).

Case 1: $\mathbf{W[|y|+k+4\ldots|y|+k+|w|+3]=w}$.

In this case $W[|y|+k+3]=1$, then necessarily by $(***)$ $U[|x|+k+3]=1$. By $(*)$, and since $U[|x|+k+3\ldots|x|+k+|u|+2]=u$, this implies that $U[|x|+k+4]=1$. Therefore by $(***)$, necessarily $W[|y|+k+4]=1$. Since $W[|y|+k+4\ldots|y|+k+|w|+3]=w$, and by $(**)$, this implies that $w=11.101$, otherwise $W[|y|+k+4]=0$, which leads to a contradiction with the fact that $U[|x|+k+4]=1$.

This implies by $(***)$, that $U[|x|+k+3\ldots|x|+k+8]=111101$. Therefore by $(*)$, necessarily $U[|x|+k+9]=0$. However by construction of W, since $W[|y|+k+4\ldots|y|+k+|w|+3]=w$, and since $|w|=5$, we have $W[|y|+k+9\ldots|y|+k+13]=w$. This implies that $W[|y|+k+9]=1$ since $w=11.101$, which leads to a contradiction with $(***)$ since $U[|x|+k+9]=0$.

Case 2: $\mathbf{W[|y|+k+2\ldots|y|+k+|w|+1]=w}$.

In this case, since $W[|y|+k+2]=0$, this implies by $(**)$ that $W[|y|+k+3]=0$. Therefore by $(***)$ we have $U[|x|+k+3]=0$. Hence, since $U[|x|+k+3\ldots|x|+k+|u|+2]=u$, then by $(*)$, we have $u=00.010$, otherwise $U[|x|+k+3]=1$ which leads to a contradiction with the fact that $W[|y|+k+3]=0$.

This implies by $(***)$, that $W[|y|+k+2\ldots|y|+k+7]=000010$. Therefore by $(**)$, necessarily $W[|y|+k+8]=1$. However by construction of U, since $U[|x|+k+3\ldots|x|+k+|u|+2]=u$, and since $|u|=5$, we have $U[|x|+k+8\ldots|x|+k+12]=u$ which implies that $U[|x|+k+8]=0$ since $u=00.010$, which leads to a contradiction with $(***)$ since $W[|y|+k+8]=1$.

□

10.3.3 Tower Properties

We are now able to state a set of lemmas that show some interesting technical properties of towers under specific assumptions during the execution of our algorithm. These properties are extensively used in the main proof of our algorithm.

This first lemma is a preliminary result used only in the proof of the two following ones (Lemmas 10.6 and 10.7).

Lemma 10.5. *The robots of a long-lived tower $T=(S,[t_s,t_e])$ consider the same global direction at each time between the Look phase of round t_s and the Look phase of round t_e included.*

Proof. Consider a long-lived tower $T=(S,[t_s,t_e])$.

Call t_{act} the first time in $[t_s,t_e[$ at which the robots of S are edge-activated. Since T is a long-lived tower, t_{act} exists.

When executing $\mathbb{SS}\_\mathbb{PEF}\_3$, a robot can change the global direction it considers only when it is edge-activated. Moreover a robot does not change the global direction it considers if it has moved during its previous edge-activation. Besides, during the Look phase of a time t a robot considers the same global direction as the one it considers during the Move phase of time $t-1$.

Therefore, during the Look phase of time t_s the robots of S consider the same global direction. Indeed, if this was not the case; i.e., if the robots of S consider different global directions during the Move phase of time t_s-1; they necessarily move during the Move phase of time t_s-1

(otherwise T is not formed at time t_s), therefore they separate during the Move phase of time t_{act}. This leads to a contradiction with the fact that T is a long-lived tower.

Consider a time $t \in]t_s, t_e[$. If at time t the robots of S are not edge-activated, then during the Move phase of time t the robots of S do not change the global direction they consider.

T is a long-lived tower from time t_s to time t_e included. Therefore if at time $t \in]t_s, t_e[$ the robots of S are edge-activated, then, by definition of a long-lived tower, during the Move phase of time t, the robots of S consider the same global direction.

Since at time t_s the robots of S consider the same global direction, using the two previous arguments by recurrence on each time $t \in]t_s, t_e[$ and the fact that robots change the global directions they consider only during Compute phases, we can conclude that the robots of S consider the same global direction from the Look phase of time t_s to the Look phase of time t_e. □

The following lemma is used to prove, in combination with Lemmas 10.3 and 10.4, the "tower breaking" mechanism since it proves that robots of a long-lived tower synchronously invoke their GiveDirection() function after their first edge-activation.

Lemma 10.6. *For any long-lived tower $T = (S, [t_s, t_e])$, any (r_i, r_j) in S^2, and any t less or equal to t_e, we have $WeAreStuckInTheSameDirection()_{r_i,t} = WeAreStuckInTheSameDirection()_{r_j,t}$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()_{r_i,t} = IWasStuckOnMyNodeAndNowWeAreMoreRobots()_{r_j,t}$ if all robots of S have been edge-activated between t_s (included) and t (not included).*

Proof. Consider a long-lived tower $T = (S, [t_s, t_e])$. Let t_{act} be the first time in $[t_s, t_e[$ where the robots of S are edge-activated. By definition of a long-lived tower, this time exists.

By definition of a long-lived tower and by lemma 10.5, from the Look phase of time t_s to the Look phase of time t_e included, all the robots of S are on the same node and consider the same global direction. Therefore the values of their respective predicates $NumberOfRobotsOnNode()$, $ExistsEdge(dir, current)$, $ExistsEdge(\overline{dir}, current)$, and $ExistsAdjacentEdge()$ are identical from the Look phase of time t_s to the Look phase of time t_e included.

When executing $\mathbb{SS}\_\mathbb{PEF}\_3$, a robot updates its variables $NumberRobotsPreviousEdgeActivation$ and $HasMovedPreviousEdgeActivation$ respectively with the values of its predicates $NumberOfRobotsOnNode()$ and $ExistsEdge(dir, current)$, only during Compute phases of times where it is edge-activated. By the observation made at the previous paragraph, this implies that from the Compute phase of time t_{act} to the Look phase of time t_e included, the robots of S have the same values for their variables $NumberRobotsPreviousEdgeActivation$ and $HasMovedPreviousEdgeActivation$.

Then, by construction of the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$, the lemma is proved.

□

From Lemmas 10.6, 10.3, and 10.4, we can deduce the following corollary stating that there exists a time from which the predicate $WeAreStuckInTheSameDirection()$ of the robots of any infinite long-lived tower is always false. This corollary is useful to prove that our algorithm solves the perpetual exploration problem in the particular case of the presence of an infinite long-lived tower. Indeed, it implies that, if there is an infinite long-lived tower, then it exists a time from which its robots are never stuck forever. Note that, in the particular case where the long-lived tower is composed of the three robots of the system, this implies that the 3 robots never change the direction they consider (since these robots cannot have their predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ true) making them able to perpetually explore the highly dynamic ring.

Corollary 10.1. *For any long-lived tower $T = (S, [t_s, t_e])$, if $t_e = +\infty$ then the predicates $WeAreStuckInTheSameDirection()$ of the robots of S cannot be infinitely often true.*

Proof. First, note that if two robots possess two distinct identifiers, then their transformed identifiers are also distinct.

Consider a long-lived tower $T = (S, \theta)$ with $\theta = [t_s, +\infty[$.

Call $t_{act} \geq t_s$ the first time after t_s where the robots of S are edge-activated. By definition of a long-lived tower, t_{act} exists. By Lemma 10.6, after time t_{act}, the robots of S consider the same values of predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNode-AndNowWeAreMoreRobots()$.

Assume by contradiction that after t_{act} the predicates $WeAreStuckInTheSameDirection()$ of the robots of S are infinitely often true. Then by construction of $\mathbb{SS}\_\mathbb{PEF}\_3$, after time t_{act}, all the robots of S call the function GiveDirection() infinitely often and at the same instants of times.

If among the robots of S two have the same chirality, to keep forming T they need to consider the same values of bits each time the function GiveDirection() is called. Here the robots have to consider the same values of bits infinitely often (since the two robots call the function GiveDirection() infinitely often). Each time a robot executes the function GiveDirection() it reads the bit next (in a round robin way) to the bit read during its previous call to the function GiveDirection(). Call i_1 and i_2 the two respective transformed identifiers of two robots forming T such that these two robots possess the same chirality. By the previous observations, to keep forming T, i_1^ω and i_2^ω must share an infinite common factor. However according to Lemma 10.3 this is not possible. Therefore there exists a time t_{end} at which these two robots consider two different bits. When the robots call the function GiveDirection(), they are edge-activated (by definition of the predicate $WeAreStuckInTheSameDirection()$), therefore at time t_{end}, T is broken.

Similarly, if among the robots of S two have not the same chirality, to keep forming T they need to consider different values of bits each time the function GiveDirection() is called. Here the robots have to consider different values of bits infinitely often (since the two robots call the function GiveDirection() infinitely often). Each time a robot executes the function GiveDirection() it reads the next bit (in a round robin way) of the bit read during its previous call to the function GiveDirection(). Call j_1 and j_2 the two respective transformed identifiers of two robots forming T such that these two robots possess a different chirality. By the previous observations, to keep forming T, j_1^ω must possess an infinite suffix S such that an infinite suffix of j_2^ω is equal to $\overline{S}$. This is equivalent to say that j_1^ω and $\overline{j_2^\omega}$ must possess an infinite common factor. However according to Lemma 10.4 this is not possible. Therefore there exists a time t_{end} at which these two robots consider two identical bits. When the robots call the function GiveDirection(), they are edge-activated, therefore at time t_{end}, T is broken.

Hence in both cases the long-lived tower T is broken, which leads to a contradiction with the fact that $\theta = [t_s, +\infty[$. □

The next lemma states one of the more fundamental properties of our algorithm: If there exists an eventual missing edge, then it is not possible for all the robots to be stuck forever on one or both of the extremities of this edge.

Lemma 10.7. *If there exists an eventual missing edge, then all long-lived towers have a finite duration.*

Proof. Consider that there exists an edge e of $\mathcal{G}$ which is missing forever from time $t_{missing}$. Consider the execution from time $t_{missing}$.

Call u and v the two adjacent nodes of e, such that v is the adjacent node of u in the clockwise direction.

By contradiction assume that there exists a long-lived tower $T = (S, \theta)$ such that $\theta = [t_s, +\infty[$. Exactly 3 robots are executing $\mathbb{SS}\_\mathbb{PEF}\_3$, so $|S|$ is either equal to 2 or 3. We want to prove that all the robots of T have their predicates $WeAreStuckInTheSameDirection()$ infinitely often true. By contradiction, assume that there exists a robot r_i of S, such that it exists a

time t_i in θ such that for all time t greater or equal to t_i its predicate $WeAreStuckInTheSameDirection()$ is false.

Call $t_{act} \geq t_s$, the first time after time t_s where the robots are edge-activated. Since T is a long-lived tower, t_{act} exists. By Lemma 10.6, from time $t_{act}+1$ the robots of S possess the same values of predicates $WeAreStuckInTheSameDirection()$. By assumption of contradiction, from time $t_{false} = max\{t_{act}+1, t_i\}$ the predicates $WeAreStuckInTheSameDirection()$ of all the robots of S are false.

We recall that by definition of a long-lived tower and by Lemma 10.5 all the robots of S are on the same node and consider the same global direction from the Look phase of time t_s to the Look phase of time t_e included.

Case 1: $|\mathbf{S}| = 3$. From time t_s the predicates $NumberOfRobotsOnNode()$ of the robots of S are equal to 3. When executing $\mathbb{SS}\_\mathbb{PEF}\_3$, a robot updates its variables $NumberRobotsPreviousEdgeActivation$ with the value of its predicate $NumberOfRobotsOnNode()$, only during Compute phases of times where it is edge-activated. Therefore, from time t_{false}, the robots of S have their variables $NumberRobotsPreviousEdgeActivation$ equal to 3. Hence, from time t_{false} their predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false, since the condition $NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation$ is false.

Since from time t_{false}, the predicates $WeAreStuckInTheSameDirection()$ of the robots of S are also false, then from time t_{false} the robots of S always consider the same global direction.

Without lost of generality, assume that, from time t_{false}, the robots of S consider the clockwise direction. All the edges of $\mathcal{G}$ except e are infinitely often present, therefore the robots of S reach node u in finite time. However e is missing forever, hence in finite time, the predicates $WeAreStuckInTheSameDirection()$ of all the robots are true. This leads to a contradiction.

Case 2: $|\mathbf{S}| = 2$. Assume, without lost of generality, that T is formed of r_1 and r_2.

If, after t_{false}, the 2-long-lived tower does not meet $\mathbf{r_3}$, then by similar arguments as the one used for the case 1 we prove that there is a contradiction.

Now consider the case where the 2-long-lived tower meets $\mathbf{r_3}$. If at a time $t' > t_{false}$, the robots of S meet r_3 it is either because the two entities (the tower and r_3) move during the Move phase of time $t'-1$ while considering two opposed global directions or because the two entities consider the same global direction but one of the entities cannot move (an edge is missing in its direction) during the Move phase of round $t'-1$. Let $t'_{act} \geq t'$ be the first time after time t' included where the three robots are edge-activated. All the edges of $\mathcal{G}$ except e are infinitely often present therefore t'_{act} exists. In both cases, thanks to the update at time $t'-1$ of the variables $HasMovedPreviousEdgeActivation$ and $NumberRobotsPreviousEdgeActivation$ of the robots, during the Move phase of time t'_{act} the robots of the two entities consider opposed global directions. The two entities separate them during the Move phase of this time. Moreover, from this separation, as long as r_3 is alone on its node it does not change the global direction it considers. Similarly, from this separation, as long as the robots of S do not meet r_3, their predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false, and since from time t_{false} their predicates $WeAreStuckInTheSameDirection()$ are false, they do not change the global direction they consider.

Hence, in finite time after time t'_{act} the two entities are located respectively on the two extremities of e. However e is missing forever, therefore in finite time, the predicates $WeAreStuckInTheSameDirection()$ of the robots of T are true. This leads to a contradiction.

In both cases a contradiction is highlighted. Therefore, after t_{false} all the robots of S have their predicates $WeAreStuckInTheSameDirection()$ infinitely often true. Then we can use the

contrapositive of Corollary 10.1 to prove that T is broken, which leads to a contradiction with the fact that $\theta = [t_s, +\infty[$. □

While proving the correctness of $\mathbb{SS}\_\mathbb{PEF}\_3$, we decompose the proof in multiple cases depending on the presence/absence of k-long-lived tower in the execution. The following lemma states that, in the case where there is no long-lived tower in the execution, then the execution contains only configurations with either three isolated robots or one 2-short-lived tower and one isolated robot.

Lemma 10.8. *No execution containing only configurations without a long-lived tower reaches a configuration where three robots form a tower.*

Proof. Assume that there is no long-lived tower in the execution. The robots can cross at most one edge at each round. Each node has at most 2 adjacent edges in $\mathcal{G}$. Moreover each robot considers at each instant time a direction. Assume, by contradiction that 3 robots form a tower T at a time t. Let $t' \geq t$ be the first time after time t where the robots of T are edge-activated. There is no 3-long-lived tower in the execution, therefore during the Move phase of time t', the robots of T consider two opposed global directions. However there are three robots, and two different global directions, hence, during the Move phase of time t', two robots of T consider the same global direction. Therefore there exists a 2-long-lived tower, which leads to a contradiction. □

The following lemma is a technical preliminary used in the proof of Lemmas 10.10 and 10.11.

Lemma 10.9. *In every execution, if a tower involving 3 robots is formed at time t, then at time $t-1$ a 2-long-lived tower is present in ε.*

Proof. Assume that a tower T of 3 robots is formed at time t.

First note that if there exists a 2-long-lived tower $T' = (S, [t_s, t_e])$ such that $t - 1 \in [t_s, t_e[$, it is possible for T to be formed.

Now we prove that if there is no 2-long-lived tower at time $t-1$ then T cannot be formed at time t. Assume that at time $t-1$ there is no 2-long-lived tower. Let us consider the three following cases.

Case 1: There is a tower $\mathbf{T'}$ of 3 robots at time $\mathbf{t-1}$. The tower T' must break at time $t-1$, otherwise there is a contradiction with the fact that T is formed at time t. Hence the robots of T' are edge-activated at time $t-1$. While executing $\mathbb{SS}\_\mathbb{PEF}\_3$ the robots consider a direction at each round. There are only two possible directions. Therefore, for the tower T' to break at time $t-1$, two robots of T' consider the same global direction, while the other robot of T' considers the opposite global direction. This implies that the three robots cannot be present on the same node at time t, since $n \geq 4$.

Case 2: There is a 2-short-lived tower T' at time $\mathbf{t-1}$. For the three robots to form T at time t, they must be edge-activated at time $t-1$. By definition of a 2-short-lived tower, the two robots of T' consider two opposed global directions during the Move phase of time $t-1$. Since the robots can cross at most one edge at each round, it is not possible for the three robots to be on the same node at time t, which leads to a contradiction with the fact that T is formed at time t.

Case 3: There are 3 isolated robots at time $\mathbf{t-1}$. For the three robots to form T at time t, they must be edge-activated at time $t-1$. The robots can cross at most one edge at each round. Each node has at most 2 adjacent edges present in $\mathcal{G}$. Moreover each robot considers at each instant time a direction. Therefore it is not possible for the three robots to be on the same node at time t, which leads to a contradiction with the fact that T is formed at time t.

□

The following lemma shows that our algorithm ensures that the absence of 3-long-lived tower is a closed property.

Lemma 10.10. *Every execution starting from a configuration without a 3-long-lived tower cannot reach a configuration with a 3-long-lived tower.*

Proof. Assume that $\mathcal{E}$ starts from a configuration which does not contain a 3-long-lived tower. By contradiction, let γ be the first configuration of $\mathcal{E}$ containing a 3-long-lived tower $T = (S, [t_s, t_e])$.

Let $t_{act} \geq t_s$ be the first time after time t_s where the 3 robots of T are edge-activated. By definition of a long-lived tower, t_{act} exists.

Lemma 10.9 implies that the configuration at time $t_s - 1$ contains a 2-long-lived tower. Hence, since γ contains the first 3-long-lived tower of $\mathcal{E}$, at time t_s a 2-long-lived tower and a robot meet to form T. The meeting between these two entities can happen either because both of them move in opposed global directions during the Move phase of time $t_s - 1$, or because, during the Move phase of time $t_s - 1$, the two entities consider the same global direction but one of the entities cannot move (an edge is missing in its direction). In both cases; thanks to the update of the variables $HasMovedPreviousEdgeActivation$ and $NumberRobotsPreviousEdgeActivation$ at time $t_s - 1$; during the Move phase of time t_{act} the two entities consider opposed global directions. Hence, the two entities separate during the Move phase of time t_{act}, therefore there is a contradiction with the fact that T is a 3-long-live tower. □

The following lemma is a technical lemma used commonly by Lemmas 10.12 and 10.13.

Lemma 10.11. *Consider an execution $\mathcal{E}$ without any 3-long-lived tower. If a 2-long-lived tower T is formed at a time t_s, then during the Look phase of time $t_s - 1$, a tower T' of 2 robots involving only one robot of T is present. Moreover, during the Move phase of time $t_s - 1$, the robot of T involved in T' does not move while the other robot of T moves.*

Proof. Consider an execution $\mathcal{E}$ without any 3-long-lived tower. Assume that at time t_s a 2-long-lived tower $T = (S, [t_s, t_e])$ is formed.

First note that if there exists a tower T' of 2 robots at time $t_s - 1$, such that only one robot of T' is involved in T and such that this robot does not move during the Move phase of time $t_s - 1$, then it is possible for T to be formed. Now we prove that T can be formed at time t_s only in this situation.

Assume, by contradiction, that there is no tower of 2 robots during the Look phase of time $t_s - 1$. This implies that, at time $t_s - 1$ either the three robots are involved in a 3-short-lived tower T_3 (case 1) or the three robots are isolated (case 2).

Case 1: Call t, the time of the formation of T_3. At time $t_s - 1$ the robots of T_3 are edge-activated, otherwise T cannot be formed at time t_s. By definition of a 3-short-lived tower, during the Move phase of time $t_s - 1$ the robots of T_3 separate. While executing $\mathbb{SS}\_\mathbb{PEF}\_3$, each robot considers at each instant time a direction. Therefore, during the Move phase of time $t_s - 1$ two robots of T_3 consider the same global direction. These two robots are still on the same node at time t_s, hence only these two robots can be involved in T. However, since these two robots are on the same node at least from time t and consider the same global direction when they are edge-activated during the Move phase of time $t_s - 1$, there is a contradiction with the fact that T is formed at time t_s.

Case 2: At time $t_s - 1$ the robots of T must be edge-activated, otherwise there is a contradiction with the fact that T starts at time t_s.

Since there is no long-lived tower at time $t_s - 1$ then by Lemma 10.9, at time t_s it is not possible to have a tower of 3 robots. Then since at time t_s, T is formed, it exists at time t_s

a tower of 2 robots. For two robots to form a tower at time t_s, during the Move phase of time $t_s - 1$, they either both move while considering two opposed global directions or they consider the same global direction but one of the robots cannot move (an edge is missing in its direction). In both cases, thanks to the update of their variables $NumberRobots\text{-}PreviousEdgeActivation$ and $HasMovedPreviousEdgeActivation$ during the Compute phase of time $t_s - 1$, during the Move phase of the first time greater or equal to t_s where these two robots are edge-activated, they consider opposed global directions and separate them. Therefore there is a contradiction with the fact that T is a 2-long-lived tower starting at time t_s.

Therefore there exists a tower of 2 robots T' during the Look phase of time $t_s - 1$. Now assume, by contradiction that the two robots of T' are involved in T. If T' is a 2-long-lived tower then during the Move phase of time $t_s - 1$ the two robots of T' are edge-activated and consider two opposed global directions, otherwise there is a contradiction with the fact that T starts at time t_s. If T' is a 2-short-lived tower then during the Move phase of time $t_s - 1$ the two robots of T' are edge-activated (otherwise T cannot be a 2-long-lived tower), and they consider two opposite global directions (by definition of a 2-short-lived tower).

A robot can cross only one edge at each instant time. Since $n \geq 4$ whatever the situation (only one of the robots of T moves or both of the robots of T move during the Move phase of time $t_s - 1$) the two robots of T cannot be again on the same node at time t_s. In conclusion, only one robot of T' is involved in T.

Finally, assume by contradiction, that during the Move phase of time $t_s - 1$, either both the robots of T move (in this case, during the Move phase of time $t_s - 1$ the two robots consider two opposed global directions otherwise they cannot meet to form T) or only the robot of T involved in T' moves while the other robot of T does not move (in this case, during the Move phase of time $t_s - 1$ the two robots consider the same global direction otherwise they cannot meet to form T). In both cases, thanks to the update of the variables $HasMovedPreviousEdgeActivation$ and $NumberRobotsPreviousEdgeActivation$ during the Compute phase of time $t_s - 1$, during the Move phase of the first time after time t_s where the robots of T are edge-activated, they consider two opposed global directions. Therefore there is a contradiction with the fact that T is a 2-long-lived tower starting at time t_s. □

The next two lemmas show that the whole ring is visited between two consecutive formations of 2-long-lived towers if these two towers satisfy some properties. Intuitively, this fundamental property of our algorithm is mainly due to the fact that the robots of a long-lived tower, when they detect they are stuck, wait one edge-activation before trying to break this tower.

Indeed, if this mechanism of "tower breaking" was not delayed from one edge-activation, then it is possible to not solve the perpetual exploration problem since a 2-long-lived tower T may then be formed just after another one T' is broken, without having the whole ring explored at least once by the robots. The formation of T is due to the meeting between a robot r executing the sentinel/visitor scheme and a robot that was involved in T'. As long as r is involved in T, it cannot visit anymore the nodes of the ring since it is involved in a 2-long-lived tower, and since the robots of this tower may be stuck on their node and hence try to break it. While being broken, one of the robots of T can meet the robot not involved in it (and which is therefore executing the sentinel/visitor scheme), creating another 2-long-lived tower, *etc.* Therefore, without this delay, it is possible to prevent the robots to execute the sentinel/visitor scheme and hence to explore the ring.

Lemma 10.12. *Consider an execution $\mathcal{E}$ without any 3-long-lived tower but containing a 2-long-lived tower $T = (S, [t_s, t_e])$. If there exists another 2-long-lived tower $T' = (S', [t'_s, t'_e])$, with $t'_s > t_e + 1$ and such that T' is the first 2-long-lived tower after T in $\mathcal{E}$, then all the nodes of $\mathcal{G}$ have been visited by at least one robot between time t_e and time $t'_s - 1$.*

Proof. Consider an execution $\mathcal{E}$ without any 3-long-lived tower but containing a 2-long-lived tower $T = (S, [t_s, t_e])$. Assume that there exists another 2-long-lived tower $T' = (S', [t'_s, t'_e])$, with $t'_s > t_e + 1$ and such that T' is the first 2-long-lived tower after T in $\mathcal{E}$.

Since by assumption there is no long-lived tower between the Look phase of time $t_e + 1$ and the Look phase of time $t'_s - 1$ included, then by Lemma 10.8, from the Look phase of time $t_e + 1$ to the Look phase of time $t'_s - 1$ included, if some robots meet they only form 2-short-lived towers. Therefore, by Lemma 10.11, at time $t'_s - 1$ there exists a 2-short-lived tower T_{short}.

To form T', by Lemma 10.11, the configuration C reached is such that T_{short} and the robot of T' not involved in T_{short} are on two adjacent nodes, the adjacent edge to the location of T_{short} in the global direction d is missing at time $t'_s - 1$, and the two robots of T' are edge-activated and consider the global direction d during the Move phase of time $t'_s - 1$. During the Move phase of time t_e the configuration C' is such that the two robots of T are on the same node considering two opposed global directions. Moreover, from the Look phase of time $t_e + 1$ to the Look phase of time $t'_s - 1$ included, if two robots meet they separate once they are edge-activated considering two opposed global directions. Besides, while executing $\mathbb{SS}\_\mathbb{PEF}\_3$, a robot does not change the global direction it considers if it is isolated. All this implies that to reach C from C' all the nodes of $\mathcal{G}$ have been visited by at least one robot between time t_e and time $t'_s - 1$. □

Lemma 10.13. *Consider an execution $\mathcal{E}$ without any 3-long-lived tower, and let $T_i = (S_i, [t_{s\_i}, t_{e\_i}])$ be the i^{th} 2-long-lived tower of $\mathcal{E}$, with $i \geq 2$. If $T_{i+1} = (S_{i+1}, [t_{s\_i+1}, t_{e\_i+1}])$ exists and satisfies $t_{s\_i+1} = t_{e\_i} + 1$, then all the nodes of $\mathcal{G}$ have been visited by at least one robot between time $t_{s\_i} - 1$ and time $t_{s\_i+1} - 1$.*

Proof. Consider an execution $\mathcal{E}$ without any 3-long-lived tower but containing a 2-long-lived tower $T_i = (S_i, [t_{s\_i}, t_{e\_i}])$, with $i \geq 2$. Assume that there exists another 2-long-lived tower $T_{i+1} = (S_{i+1}, [t_{s\_i+1}, t_{e\_i+1}])$, with $t_{s\_i+1} = t_{e\_i} + 1$. By Lemma 10.11, to form T_{i+1}, a tower of 2 robots involving only one robot of T_{i+1} must be present at time $t_{s\_i+1} - 1$. Moreover T_i is a tower of 2 robots which is present in $\mathcal{G}$ from time t_{s_i} to time $t_{s\_i+1} - 1$. Therefore $S_{i+1} \neq S_i$.

To form T_i, by Lemma 10.11, the configuration C reached at time $t_{s\_i} - 1$ is such that there is a tower T of 2 robots involving only one robot of T_i and the other robot of T_i which are on two adjacent nodes.

Similarly, by Lemma 10.11, and since $t_{s\_i+1} = t_{e\_i} + 1$, to form T_{i+1}, the configuration C' reached at time $t_{s\_i+1} - 1$ is such that T_i and the robot of T_{i+1} not involved in T_i are on two adjacent nodes, the adjacent edge to the location of T_i in the global direction d is missing at time $t_{s\_i+1} - 1$, and the two robots of T_{i+1} are edge-activated and consider the global direction d during the Move phase of time $t_{s\_i+1} - 1$. Moreover, since there is no 3-long-lived tower in $\mathcal{E}$, from the Look phase of time t_{s_i} to the Look phase of time $t_{s\_i+1} - 1$ included, if T_i meets the other robot of the system, they form a 3-short-lived tower and hence they separate once they are edge-activated considering two opposed global directions. Besides, while executing $\mathbb{SS}\_\mathbb{PEF}\_3$, a robot does not change the global direction it considers if it is isolated. All this implies that to reach C' from C all the nodes of $\mathcal{G}$ have been visited by at least one robot between time $t_{s\_i} - 1$ and time $t_{s\_i+1} - 1$. □

10.3.4 Correctness Proof

Upon establishing all the above properties of towers, we are now ready to state the main lemmas of our proof. Each of these three lemmas below shows that after time t_{max} our algorithm performs the perpetual exploration in a self-stabilizing way for some specific subclasses of $\mathcal{COT}$ rings that form a partition of $\mathcal{COT}$.

Lemma 10.14. $\mathbb{SS}\_\mathbb{PEF}\_3$ *satisfies the perpetual exploration problem in $\mathcal{ST}$ rings of arbitrary size under $\mathcal{M}_{self-stabilizing\ exploration}$ using three robots.*

Proof. Assume that $\mathcal{G}$ is a $\mathcal{ST}$ ring. While executing $\mathbb{SS}\_\mathbb{PEF}\_3$, a robot considers a direction at each round. Moreover, a robot does not change the global direction it considers if its variable $HasMovedPreviousEdgeActivation$ is true. The variables of a robot are updated during Compute phases of times where it is edge-activated. Since $\mathcal{G}$ is a $\mathcal{ST}$ ring, this implies that in each round all the robots are edge-activated and are able to move whatever the direction they consider. So, after t_{max} their variables $HasMovedPreviousEdgeActivation$ are always true. Hence, the robots never change their directions.

As (i) the robots have a stable direction, (ii) they always consider respectively the same global direction, and (iii) there always exists an adjacent edge to their current locations in the global direction they consider, the robots move infinitely often in the same global direction. Moreover, as $\mathcal{G}$ has a finite size, this implies that all the robots visit infinitely often all the nodes of $\mathcal{G}$. □

Lemma 10.15. $\mathbb{SS}\_\mathbb{PEF}\_3$ *satisfies the perpetual exploration problem in* $\mathcal{RE}\backslash\mathcal{ST}$ *rings of arbitrary size under* $\mathcal{M}_{self-stabilizing\ exploration}$ *using three robots.*

Proof. Assume that $\mathcal{G}$ is a $\mathcal{RE}\backslash\mathcal{ST}$ ring. Let us study the following cases.

Case 1: There exists at least one 3-long-lived tower in $\mathcal{E}$.

Case 1.1: One of the 3-long-lived towers of $\mathcal{E}$ has an infinite duration.

Denote by $T = (S, [t_s, +\infty[)$ the 3-long-lived tower of $\mathcal{E}$ that has an infinite duration. Call $t_{act} \geq t_s$ the first time after time t_s where the robots of T are edge-activated. By definition of a long-lived tower, t_{act} exists. The variables of a robot are updated during Compute phases of times where it is edge-activated. Therefore, since there are three robots in the system, from time $t_{act} + 1$, the condition "$NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation$" is false for the three robots of T. Therefore from time $t_{act} + 1$ the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of each robot of T is false.

By Corollary 10.1, eventually, the predicates $WeAreStuckInTheSameDirection()$ of the robots of T are always false, otherwise T is broken in finite time, which leads to a contradiction.

Since eventually the predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ and $WeAreStuckInTheSameDirection()$ of the robots of T are always false, then eventually they never change the global direction they consider. $\mathcal{G}$ is a $\mathcal{RE}$ ring, therefore there exists infinitely often an adjacent edge to the location of T in the global direction considered by the robots of T, then the robots are able to move infinitely often in the same global direction. Moreover, as $\mathcal{G}$ has a finite size, all the robots visit infinitely often all the nodes of $\mathcal{G}$.

Case 1.2: Any 3-long-lived tower of $\mathcal{E}$ has a finite duration.

By Lemma 10.10, once a 3-long-lived tower is broken, it is impossible to have another 3-long-lived tower in $\mathcal{E}$. Then, $\mathcal{E}$ admits an infinite suffix that matches either case 2 or 3.

Case 2: There exists at least one 2-long-lived tower in $\mathcal{E}$.

Case 2.1: There exists a finite number of 2-long-lived towers in $\mathcal{E}$.

Let $T' = (S', [t'_s, t'_e])$ be the last 2-long-lived tower of $\mathcal{E}$.

There is no 3-long-lived tower in $\mathcal{E}$ at time t'_s (otherwise Case 1 is considered), hence by Lemma 10.10 there is no 3-long-lived tower in $\mathcal{E}$. Moreover, if T' has a finite duration, then $\mathcal{E}$ admits an infinite suffix with no long-lived tower, hence matching case 3.

Otherwise, (i.e., T' has an infinite duration), as in Case 1.1, the robots of T' eventually have their predicates $WeAreStuckInTheSameDirection()$ always false, otherwise, T'

is broken in finite time. Let t_{false} be the time from which the robots of T' have their predicates $WeAreStuckInTheSameDirection()$ always false. After time t_{false}, the only case when the robots of T' change the global direction they consider, is when they meet the third robot of the system.

Case 2.1.1: The robots of T′ meet the third robot finitely often.

After the time when the last tower of 3 robots is broken, the robots of T' have their predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ always false. Let t_{break} be the time when the last tower of 3 robots is broken. From time $t = max\{t_{break}, t_{false}\} + 1$ the robots of T' have their predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ and $WeAreStuckInTheSameDirection()$ always false, therefore they never change the global direction they consider. Since $\mathcal{G}$ is a $\mathcal{RE}$ ring, there is infinitely often an adjacent edge to the location of T' in the direction considered by the robots of T'. This implies that they are able to move infinitely often in the same global direction. Moreover, as $\mathcal{G}$ has a finite size, this implies that all the nodes of $\mathcal{G}$ are visited infinitely often.

Case 2.1.2: The robots of T′ meet the third robot infinitely often.

Consider the execution after time t_{false}. The robot not involved in T' does not change its direction while it is isolated. Similarly, the robots of T' maintain their directions at least until they meet the third robot. Moreover, when the robots of T' meet the third robot of the system, they form a 3-short-lived tower. Therefore once they are edge-activated, they separate them considering opposed global directions. Then, we can deduce that all the nodes of $\mathcal{G}$ are visited between two consecutive meetings of T' and the third robot. As T' and the third robot infinitely often meet, all the nodes of $\mathcal{G}$ are infinitely often visited.

Case 2.2: There exist an infinite number of 2-long-lived towers in $\mathcal{E}$.

By Lemmas 10.12 and 10.13, we know that between two consecutive 2-long-lived towers (from the second one), all the nodes of $\mathcal{G}$ are visited. As there is an infinite number of 2-long-lived towers, the nodes of $\mathcal{G}$ are infinitely often visited.

Case 3: There does not exist a long-lived tower in $\mathcal{E}$.

Then, we know, by Lemma 10.8, that $\mathcal{E}$ contains only configurations with either three isolated robots or one 2-short-lived tower and one isolated robot.

We want to prove the following property. If during the Look phase of time t, a robot r is located on a node u considering the global direction gd, then there exists a time $t' \geq t$ such that, during the Look phase of time t', a robot is located on the node v adjacent to u in the global direction gd and considers the global direction gd.

Let $t" \geq t$ be the smallest time after time t where the adjacent edge of u in the global direction gd is present in $\mathcal{G}$. As all the edges of $\mathcal{G}$ are infinitely often present, $t"$ exists.

(i) If r crosses the adjacent edge of u in the global direction gd during the Move phase of time $t"$, then the property is verified.

(ii) If r does not cross the adjacent edge of u in the global direction gd during the Move phase of time $t"$, this implies that r changes the global direction it considers during the Look phase of time t. While executing $\mathbb{SS}\_\mathbb{PEF}\_3$, a robot can change the global direction it considers only during Compute phases of times where it is edge-activated and involved in a tower. Let $t_{act} \geq t$ be the first time after time t such that during the Move phase of time t_{act}, r does not consider the global direction gd. Let r' be the robot involved in a tower with r at time t_{act}. Since there are only 2-short-lived towers in the execution, the two robots r and r' consider two opposed global directions during the Move phase of time t_{act}. Therefore during the Move phase of time t_{act}, r' is on node u considering the global direction gd. By

applying case (ii) by recurrence, we can say that from the Move phase of time t to the Move phase of time t" there always exists a robot on node u considering the global direction gd. Therefore during the Move phase of time t" a robot moves on node v. Since the robot does not change the global direction they consider during Look phases, during the Look phase of time t" $+ 1$ this robot still considers the global direction gd.

This proves the property. By applying recurrently this property to any robot, we prove that all the nodes of $\mathcal{G}$ are infinitely often visited.

Thus, we obtain the desired result in every case. □

Lemma 10.16. $\mathbb{SS}\_\mathbb{PEF}\_3$ *satisfies the perpetual exploration problem in* $\mathcal{COT}\backslash\mathcal{RE}$ *rings of arbitrary size under* $\mathcal{M}_{self-stabilizing\ exploration}$ *using three robots.*

Proof. Consider that $\mathcal{G}$ is a $\mathcal{COT}\backslash\mathcal{RE}$ ring. This implies that there exists exactly one eventual missing edge e in $\mathcal{G}$. Denote by $\mathcal{E}'$ the maximal suffix of $\mathcal{E}$ in which the eventual missing edge never appears. Let $t_{missing}$ the time after which e never appears again. Let us study the following cases.

Case 1: There exists at least one 3-long-lived tower in $\mathcal{E}'$.

According to Lemma 10.7, this 3-long-liver tower is broken in finite time. Moreover, once this tower is broken, according to Lemma 10.10, it is impossible to have a configuration containing a 3-long-lived tower. Then, $\mathcal{E}'$ admits an infinite suffix that matches either case 2 or 3.

Case 2: There exists at least one 2-long-lived tower in $\mathcal{E}'$.

Case 2.1: There exists a finite number of 2-long-lived towers in $\mathcal{E}'$.

According to Lemma 10.7, the last 2-long-lived tower is broken in finite time. Since by Lemma 10.10, it cannot exist 3-long-lived tower in $\mathcal{E}'$, then $\mathcal{E}'$ admits an infinite suffix with no long-lived tower hence matching Case 3.

Case 2.2: There exist an infinite number of 2-long-lived towers in $\mathcal{E}'$.

By Lemmas 10.12 and 10.13, we know that between two consecutive 2-long-lived towers (from the second one), all the nodes of $\mathcal{G}$ are visited. As there are an infinite number of 2-long-lived towers, all the nodes of $\mathcal{G}$ are infinitely often visited.

Case 3: There does not exist a long-lived tower in $\mathcal{E}'$.

By Lemma 10.8, all configurations in $\mathcal{E}'$ contain either three isolated robots or one 2-short-lived tower and one isolated robot.

(1) We want to prove the following property. If during the Look phase of a time t in $\mathcal{E}'$, a robot considers a global direction gd and is located on a node at a distance $d \neq 0$ in G (G is the footprint of $\mathcal{G}$) from the extremity of e in the global direction gd, then it exists a time $t' \geq t$ such that, during the Look phase of time t', a robot is on a node at distance $d - 1$ in G from the extremity of e in the global direction gd and considers the global direction gd. Let v be the adjacent node of u in the global direction gd.

Let t" $\geq t$ be the smallest time after time t where the adjacent edge of u in the global direction gd is present in $\mathcal{G}$. As all the edges of $\mathcal{G}$ except e are infinitely often present and as u is at a distance $d \neq 0$ in G from the extremity of e in the global direction gd, then the adjacent edge of u in the global direction gd is infinitely often present in $\mathcal{G}$. Hence, t" exists.

(i) If r crosses the adjacent edge of u in the global direction gd during the Move phase of time t", then the property is verified.

(ii) If r does not cross the adjacent edge of u in the global direction gd during the Move phase of time t", this implies that r changes the global direction it considers during the Look phase of time t. While executing $\mathbb{SS}\_\mathbb{PEF}\_3$, a robot can change the global direction it considers only during Compute phases of times where it is edge-activated and involved in a tower. Let $t_{act} \geq t$ be the first time after time t such that during the Move phase of time t_{act}, r does not consider the global direction gd. Let r' be the robot involved in a tower with r at time t_{act}. Since there are only 2-short-lived towers in the execution, the two robots r and r' consider two opposed global directions during the Move phase of time t_{act}. Therefore during the Move phase of time t_{act}, r' is on node u considering the global direction gd. By applying case (ii) by recurrence, we can say that from the Move phase of time t to the Move phase of time t" there always exists a robot on node u considering the global direction gd. Therefore during the Move phase of time t" a robot moves on node v. Since the robot does not change the global direction they consider during Look phases, during the Look phase of time $t" + 1$ this robot still considers the global direction gd.

This proves the property.

(2) We now want to prove that there exists a time $t_{reachExtremities}$ in $\mathcal{E}'$ from which one robot is forever located on each extremity of e pointing to e.

First, we want to prove that a robot reaches one of the extremities of e in a finite time after $t_{missing}$ and points to e at this time. If it is not the case at time $t_{missing}$, then there exists at this time a robot considering a global direction gd and located on a node u at distance $d \neq 0$ in G from the extremity of e in the global direction gd. By applying d times the property (1), we prove that, during the Look phase of a time $t_{reach} \geq t_{missing}$, a robot (denote it r) reaches the extremity of e in the global direction gd from u (denote it v and let v' be the other extremity of e), and that this robot considers the global direction gd during the Look phase of time t_{reach}.

Then, we can prove that from time t_{reach} there always exists a robot on node v considering the global direction gd. Indeed, note that no robot can cross e in the global direction gd from time t_{reach} since e is missing from time $t_{missing}$. Moreover while executing $\mathbb{SS}\_\mathbb{PEF}\_3$, a robot can change the global direction it considers only during Compute phases of times where it is edge-activated and involved in a tower. Therefore if at a time $t_{change} \geq t_{missing}$, r changes the global direction it considers at time t_{reach} this is because it is involved in a tower. Since there are only 2-short-lived towers in the execution, at time t_{change}, r is involved in a tower with a robot r', and r and r' consider two opposed global directions during the Move phase of time t_{change}. Therefore during the Move phase of time t_{change}, r' is on node v considering the global direction gd. By applying this argument by recurrence, we can say that from time t_{reach} there always exists a robot on node v considering the global direction gd.

Now we prove that this is also true for the extremity v' of e. If there exists at time t_{reach} a robot on node v' considering the global direction $\overline{gd}$, or if it exists a robot considering the global direction $\overline{gd}$ on a node u' at distance $d \neq 0$ in G from v' in the global direction $\overline{gd}$, then by using similar arguments as the one used for v, we can prove the property (2). If this is not the case, this implies that at time t_{reach} all the robots consider the global direction gd. Then in finite time (after time t_{reach}) by the property (1), a robot reaches node v. Since from time t_{reach} there is always a robot on node v, there is a 2-short-lived tower formed. Then by definition of a 2-short-lived tower, there exists a time at which one of the robots of this tower considers the global direction gd while the other considers the global direction $\overline{gd}$. Then we can use the same arguments as the one used previously to prove the property (2).

(3) It stays to prove that in the Case 3 all the nodes are infinitely often visited. We know that from time $t_{reachExtremities}$ one robot is forever located on each extremity of e point-

ing to e. Call r" the robot that is not on node v (resp. v') and pointing to e at time $t_{reachExtremities}$. Assume, without lost of generality, that at time $t_{reachExtremities}$, r" is on node u' and considers the global direction gd. Then by applying recurrently the property (1) we can prove that, in finite time, all the nodes between the current node of r" at time $t_{reachExtremities}$ and v in the global direction gd are visited and that r" reaches v. Call $t'_{act} \geq t_{reachExtremities}$, the first time after time $t_{reachExtremities}$ where there are two robots on node v that are edge-activated. At time t'_{act}, the robot that is on node v and pointing to e at time $t_{reachExtremities}$ changes the global direction it considers (hence considers $\overline{gd}$) by construction of $\mathbb{SS}\_\mathbb{PEF}\_3$ and since the tower formed is a 2-short-lived tower.

We can then repeat this reasoning (with v and v' alternatively in the role of u' and with v' and v alternatively in the role of v) and prove that all nodes of $\mathcal{G}$ are infinitely often visited.

Thus, we obtain the desired result in every case. □

To conclude the proof, first note that even if the robots can start in a non coherent state, it exists a time t_{max} from which all the robots of the system are in a coherent state (by Lemma 10.2). Then it is sufficient to observe that a $\mathcal{COT}$ ring is by definition either a $\mathcal{ST}$ ring, a $\mathcal{RE}\backslash\mathcal{ST}$ ring, or a $\mathcal{COT}\backslash\mathcal{RE}$ ring. As we prove the correctness of our algorithm from the time the robots are in a coherent state in these three cases in Lemmas 10.14, 10.15, and 10.16 respectively, we can claim the following final result.

Theorem 10.3. $\mathbb{SS}\_\mathbb{PEF}\_3$ *is a self-stabilizing perpetual exploration algorithm for* $\mathcal{COT}$ *rings of arbitrary size (greater or equal to four) under* $\mathcal{M}_{self-stabilizing\ exploration}$ *using three robots.*

10.4 Speculative Aspect of $\mathbb{SS}\_\mathbb{PEF}\_3$

In this section we prove that $\mathbb{SS}\_\mathbb{PEF}\_3$ is a speculative algorithm. As indicated in Section 8.3, the cover time of any perpetual exploration algorithm in $\mathcal{COT}$ rings is unbounded (since it is possible, in a $\mathcal{COT}$ graph, to have initially and during an arbitrary long time all the edges of the graph missing). Since, by Theorem 10.3, $\mathbb{SS}\_\mathbb{PEF}\_3$ solves the perpetual exploration problem in $\mathcal{COT}$ rings, and since its cover time in $\mathcal{COT}$ rings is unbounded, the following lemma is sufficient to prove that $\mathbb{SS}\_\mathbb{PEF}\_3$ is a speculative algorithm for the perpetual exploration problem with respect to $\mathcal{ST}$ and $\mathcal{COT}$, in function of the cover time in rounds. Indeed, in the lemma below, we prove that the cover time of $\mathbb{SS}\_\mathbb{PEF}\_3$ in $\mathcal{ST}$ rings is bounded.

Lemma 10.17. *Our algorithm* $\mathbb{SS}\_\mathbb{PEF}\_3$ *under* $\mathcal{M}_{self-stabilizing\ exploration}$*, using* 3 *robots in any* $\mathcal{ST}$ *ring of size* n *has a cover time in* $O(n)$ *rounds.*

Proof. First note that in the case of a $\mathcal{ST}$ ring, $t_{max} = 2$.

While executing $\mathbb{SS}\_\mathbb{PEF}\_3$, a robot considers a direction (right or left) at each round. Moreover, a robot does not change the global direction it considers if its variable $HasMovedPreviousEdgeActivation$ is true. The variables of a robot are updated during Compute phases of times where it is edge-activated. Since the ring is a $\mathcal{ST}$ ring, this implies that in each round all the robots are edge-activated and are able to move whatever the direction they consider. So, after t_{max} their variables $HasMovedPreviousEdgeActivation$ are always true. Hence, the robots never change their directions after this time.

As (i) the robots have a stable direction, (ii) they always consider respectively the same global direction, and (iii) there always exists an adjacent edge to their current locations in the global direction they consider, the cover time is equal to $n + t_{max}$ rounds. Since $n + 2$ is in $O(n)$, this proves the lemma. □

We now prove that the cover time of $\mathbb{SS}\_\mathbb{PEF}\_3$ is moreover asymptotically optimal when executed in $\mathcal{ST}$ rings (see Lemma 10.18). To do so, we show that the cover time of any self-stabilizing deterministic algorithm solving the perpetual exploration problem in $\mathcal{ST}$ rings of size n using $\mathcal{R}$ robots is equal to the cover time of our algorithm (proved in Lemma 10.17).

Lemma 10.18. *Any self-stabilizing deterministic algorithm under $\mathcal{M}_{self-stabilizing\ exploration}$ using $\mathcal{R}$ robots, satisfying the perpetual exploration problem in $\mathcal{ST}$ rings of size n has a cover time in $\Omega(n)$ rounds.*

Proof. Assume, by contradiction, that there exists a self-stabilizing deterministic algorithm $\mathcal{A}$ using $\mathcal{R}$ robots, satisfying the perpetual exploration problem in $\mathcal{ST}$ rings of size n, that possesses a cover time less or equals to $\tau = \lfloor n/2 \rfloor - 1$ rounds.

Consider the execution of $\mathcal{A}$ in a $\mathcal{ST}$ ring of size n such that initially all the robots are on the same node x. Call γ this initial configuration.

Call y a node at distance $\lfloor n/2 \rfloor$ of x.

Since each robot can cross at most one edge at each round, this implies that each robot can explore at most $\lfloor n/2 \rfloor - 1$ different nodes in τ rounds during the execution of $\mathcal{A}$ starting from γ. Note that all the nodes visited by a robot are consecutive nodes. Therefore, whatever the direction considered by each of the robots, none of them succeed to visit the node y during the τ first rounds of the execution of $\mathcal{A}$ starting from γ. Hence there is a contradiction with the fact that $\mathcal{A}$ is a self-stabilizing deterministic algorithm using $\mathcal{R}$ robots, satisfying the perpetual exploration problem in $\mathcal{ST}$ rings of size n and having a cover time less or equals to $\lfloor n/2 \rfloor - 1$ rounds.

This implies that there is no self-stabilizing deterministic algorithm satisfying the perpetual exploration problem in $\mathcal{ST}$ rings of size n, using $\mathcal{R}$ robots and having a cover time less or equals to $\lfloor n/2 \rfloor - 1$ rounds. Hence, since $\lfloor n/2 \rfloor - 1$ is in $\Omega(n)$, the lemma is proved. □

To conclude, as explained at the beginning of this section, and thanks to the two previous lemmas, we can deduce the following theorem.

Theorem 10.4. *$\mathbb{SS}\_\mathbb{PEF}\_3$ is a speculative algorithm under $\mathcal{M}_{self-stabilizing\ exploration}$ for the perpetual exploration problem with respect to $\mathcal{ST}$ and $\mathcal{COT}$, in function of the cover time in rounds, and its cover time in $\mathcal{ST}$ rings is asymptotically optimal.*

10.5 Sufficiency of Two Robots for $n = 3$

In this section, we present $\mathbb{SS}\_\mathbb{PEF}\_2$, a self-stabilizing algorithm solving deterministically the perpetual exploration problem in $\mathcal{COT}$ rings of size equal to three, using two robots possessing distinct identifiers.

Here, we only consider two robots. Hence, it is not possible to apply the "sentinels"/"visitor" scheme anymore (that requires three robots to be applied). In $\mathbb{SS}\_\mathbb{PEF}\_3$, a robot does not change its direction (arbitrarily initialized) while it is isolated. Here, it is not possible to apply this rule, otherwise the two robots may be stuck forever on two different extremities of an eventual missing edge. When there exists an adjacent present edge to the direction pointed to by a robot then it does not change its direction. However, when a robot is isolated and stuck (i.e., its direction points to an adjacent missing edge) it changes its direction if there exists an adjacent present edge to its location. These two behaviors are enough to permit the perpetual exploration in the case where the robots do not form a long-lived tower or in the case where they form a long-lived tower that is not stuck forever. Similarly as in $\mathbb{SS}\_\mathbb{PEF}\_3$, if the two robots form a long-lived tower and are stuck on their node, the "tower breaking" scheme is applied to break this tower.

This algorithm and its proof of correctness make use of definitions, variables, and predicates defined in Subsections 10.3.1 and 10.3.2 and of the following new predicate.

$$
\begin{aligned}
&IAmStuckAloneOnMyNode() \equiv \\
&\quad (NumberOfRobotsOnNode() = 1) \\
&\wedge \neg ExistsEdge(dir, current) \\
&\wedge ExistsEdge(\overline{dir}, current)
\end{aligned}
$$

Algorithm 10.4 $\mathbb{SS}\_\mathbb{PEF}\_2$

1: **StuckTogether** :: $WeAreStuckInTheSameDirection() \longrightarrow$ GiveDirection(); Update()
2: **StuckAlone** :: $IAmStuckAloneOnMyNode() \longrightarrow$ OppositeDirection(); Update()
3: **Update** :: true $\longrightarrow$ Update()

The pseudo-code of $\mathbb{SS}\_\mathbb{PEF}\_2$ is given in Algorithm 10.4.

Proof of correctness We now prove the correctness of this algorithm.

First, note that Lemmas 10.2, 10.3, and 10.4 are also true for $\mathbb{SS}\_\mathbb{PEF}\_2$.

To show the correctness of $\mathbb{SS}\_\mathbb{PEF}\_2$, we need to introduce some lemmas. We consider that the two robots executing $\mathbb{SS}\_\mathbb{PEF}\_2$ are r_1 and r_2. Let t_1, and t_2 be respectively the time at which the robots r_1 and r_2 are in a coherent state. Let $t_{max} = max\{t_1, t_2\}$. From Lemma 10.2, the two robots are in a coherent state from t_{max}. In the remaining of the proof, we focus on the suffix of the execution after t_{max}. The other notations correspond to the ones introduced in Section 10.3.

Lemma 10.19. *Every execution starting from a configuration without a 2-long-lived tower cannot reach a configuration with a 2-long-lived tower.*

Proof. Assume that $\mathcal{E}$ starts from a configuration which does not contain a 2-long-lived tower. By contradiction, let C be the first configuration of $\mathcal{E}$ containing a 2-long-lived tower $T = (S, [t_s, t_e])$.

Let $t_{act} \geq t_s$ be the first time after time t_s where the 2 robots of T are edge-activated. By definition of a long-lived tower, t_{act} exists.

For a 2-long-lived tower to be formed at time t_s, r_1 and r_2 must meet at time t_s. While executing $\mathbb{SS}\_\mathbb{PEF}\_2$, the two robots can meet at time t_s only because they are moving considering opposed global directions during the Move phase of time $t_s - 1$. Therefore, since the variables of a robot are updated only during Compute phases of time where it is edge-activated, during the Look phase of time t_{act}, the predicates $WeAreStuckInTheSameDirection()$ of the two robots are false (since their variables $HasMovedPreviousEdgeActivation$ are true). Moreover, during the Look phase of time t_{act} the predicates $IAmStuckAloneOnMyNode()$ of the two robots are false (since their predicates $NumberOfRobotsOnNode()$ is not equal to 1). Hence during the Move phase of time t_{act} the two robots still consider two opposed global directions. Therefore T is broken at time t_{act}, which leads to a contradiction with the fact that T is a 2-long-lived tower. This proves the lemma. □

Let t_{act1} (resp. t_{act2}) be the first time in the execution at which the robot r_1 (resp. r_2) is edge-activated. By definition, we have $t_1 = t_{act1} + 1$ and $t_2 = t_{act2} + 1$. By Lemma 10.19, if there exists a 2-long-lived tower in $\mathcal{E}$, then this 2-long-lived tower is present in the execution from time $t_0 = 0$. In this case $t_1 = t_2 = t_{max}$ and at time $t_{max} - 1$ the robots are edge-activated for the first time of the execution.

Lemma 10.20. *The robots of a long-lived tower $T = (S, [t_s, t_e])$ consider the same global direction at each time between the Look phase of round t_{max} and the Look phase of round t_e included.*

Proof. Consider a long-lived tower $T = (S, [t_s, t_e])$. We know that $t_s = t_0 = 0$, that $t_1 = t_2 = t_{max}$ and that at time $t_{max} - 1$ the robots are edge-activated for the first time of the execution. During the Move phase of time $t_{max} - 1$, the two robots consider the same global direction, otherwise there is a contradiction with the fact that T is a 2-long-lived tower.

When executing $\mathbb{SS}\_\mathbb{PEF}\_2$, a robot can change the global direction it considers only when it is edge-activated. Besides, during the Look phase of a time t a robot considers the same global direction as the one it considers during the Move phase of time $t-1$.

Consider a time $t \in [t_{max}, t_e[$. If at time t the robots of S are not edge-activated, then during the Move phase of time t the robots of S do not change the global direction they consider.

If at time t the robots of S are edge-activated, then during the Move phase of time t, since $t \neq t_e$, the robots of S consider the same global direction, otherwise there is a contradiction with the fact that T is a long-lived tower from time t_s to time t_e.

Since during the Move phase of time $t_{max}-1$ the robots of S consider the same global direction using the two previous arguments by recurrence on each time $t \in [t_{max}, t_e[$ and the fact that robots change the global directions they consider only during Compute phases, we can conclude that the robots of S consider the same global direction from the Look phase of time t_{max} to the Look phase of time t_e included. □

Lemma 10.21. *For any long-lived tower $T = (S, [t_s, t_e])$, and any $t \leq t_e$, such that the robots of S have been edge-activated twice between t_s included and t not included, we have $WeAreStuckInTheSameDirection()_{r_1,t} = WeAreStuckInTheSameDirection()_{r_2,t}$.*

Proof. Consider a long-lived tower $T = (S, [t_s, t_e])$. We know that $t_s = t_0 = 0$, that $t_1 = t_2 = t_{max}$ and that at time $t_{max}-1$ the robots are edge-activated for the first time of the execution. Assume that between t_s included and t_e not included, the robots of T are edge-activated two or more times.

By definition of a long-lived tower and by lemma 10.20, from the Look phase of time t_{max} to the Look phase of time t_e included, all the robots of S are on the same node and consider the same global direction. Therefore the values of their respective predicates $NumberOfRobotsOnNode()$, $ExistsEdge(dir, current)$ and $ExistsEdge(\overline{dir}, current)$ are identical from the Look phase of time t_{max} to the Look phase of time t_e included.

Let $t_{act} \geq t_{max}$ be the first time after t_{max} such that the robots of T are edge-activated. By assumption, t_{act} exists. When executing $\mathbb{SS}\_\mathbb{PEF}\_2$, a robot updates its variables $HasMovedPreviousEdgeActivation$ and $NumberRobotsPreviousEdgeActivation$ respectively with the values of its predicates $ExistsEdge(dir, current)$ and $NumberOfRobotsOnNode()$, only during Compute phases of times when it is edge-activated. Therefore, from the Look phase of time $t_{act}+1$ to the Look phase of time t_e included, the robots of S have the same values for their variables $HasMovedPreviousEdgeActivation$ and $NumberRobotsPreviousEdgeActivation$.

The predicate $WeAreStuckInTheSameDirection()$ depends only on the values of the variables $HasMovedPreviousEdgeActivation$, $NumberRobotsPreviousEdgeActivation$ and on the values of the predicates $NumberOfRobotsOnNode()$, $ExistsEdge(dir, current)$, and $ExistsEdge(\overline{dir}, current)$. As seen previously, all these values are identical for all the robots of S from the Look phase of time $t_{act}+1$ until the Look phase of time t_e included. This proves the lemma. □

From the Lemmas 10.21, 10.3 and 10.4, by noticing that the robots of a long-lived tower T cannot have their predicates $IAmStuckAloneOnMyNode()$ true as long as they are involved in T, we can again obtain the corollary 10.1 (the proof is not exactly the same since the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ does not exist in $\mathbb{SS}\_\mathbb{PEF}\_2$, however the proof is very similar, therefore not repeated in this section).

Theorem 10.5. $\mathbb{SS}\_\mathbb{PEF}\_2$ *is a deterministic self-stabilizing perpetual exploration algorithm for $\mathcal{COT}$ rings of size equals to* 3 *under $\mathcal{M}_{self-stabilizing\ exploration}$ using 2 robots.*

Proof. Consider that $\mathcal{G}$ is a $\mathcal{COT}$ ring of size 3. First note that even if the robots can start in a non-coherent state, by Lemma 10.2, it exists a time t_{max} from which all the robots are in a coherent state. Let us study the following cases occurring when the robots are in a coherent state.

Case 1 : There exists at least one 2-long-lived tower in $\mathcal{E}$.

By Lemma 10.19, once a 2-long-lived tower is broken, it is not possible to have again a 2-long-lived tower in $\mathcal{E}$. Therefore there exists only one 2-long-lived tower T in $\mathcal{E}$.

If T has a finite duration, then by Lemma 10.19, $\mathcal{E}$ admits an infinite suffix with no long-lived tower hence matching Case 2.

If T has an infinite duration, the robots of T eventually have their predicates $WeAreStuck\text{-}InTheSameDirection()$ always false, otherwise, by the contrapositive of Corollary 10.1, T is broken in finite time, which leads to a contradiction. Let t_{false} be the time from which the robots of T have their predicates $WeAreStuckInTheSameDirection()$ always false. After time t_{false} the robots of T never change the global direction they consider (since their predicates $IAmStuckAloneOnMyNode()$ cannot be true). Moreover, after time t_{false} there exists infinitely often an adjacent edge to the location of T in the global direction considered by the robots of T, otherwise there exists a time after t_{false} when the predicates $WeAreStuckInTheSameDirection()$ of the robots of T are true, which is a contradiction. Hence after time t_{false} the robots of T are infinitely often able to move in the same global direction. Since $\mathcal{G}$ has a finite size, all the robots visit infinitely often all the nodes of $\mathcal{G}$.

Case 2: There does not exist a long-lived tower in $\mathcal{E}$.

If there is no long-lived tower, this implies that if a tower is formed, then it is a 2-short-lived tower. By the connected-over-time assumption, each node has at least one adjacent edge infinitely often present. This implies that any short-lived tower is broken in finite time. Two cases are now possible.

Case 2.1: There exists infinitely often a 2-short-lived tower in the execution.

Note that, if a tower is formed at a time t, then the three nodes have been visited between time $t-1$ and time t. Then, the three nodes are infinitely often visited by a robot in the case where there exists infinitely often a 2-short-lived tower in the execution.

Case 2.2: There exists a time $t_{isolated}$ after which the robots are always isolated.

By contradiction, assume that there exists a time t' after which a node u is never visited. This implies that, after time $max\{t_{isolated}, t'\}$, either the robots are always switching their position or they stay on their respective nodes.

In the first case, during the Look phase of each time greater than $max\{t_{isolated}, t'\}$, the respective variables *dir* of the two robots contain the direction leading to u (since each robot previously moves in this direction). As at least one of the adjacent edges of u is infinitely often present, a robot crosses it in a finite time, that is contradictory with the fact that u is not visited after t'.

The second case implies that both adjacent edges to the location of both robots are always absent after time $t_{isolated}$ (since an isolated robot moves as soon as it is possible, by definition of the predicate $IAmStuckAloneOnMyNode()$), that is contradictory with the connected-over-time assumption.

Thus, we obtain the desired result in every case. □

10.6 Summary

In this chapter, we addressed the open question: "What is the minimal size of a swarm of self-stabilizing robots to perform perpetual exploration of highly dynamic graphs?" We give the first answer to this question by exhibiting the necessary and sufficient numbers of such robots to perpetually explore any $\mathcal{COT}$ ring. More precisely, we showed that the necessary and sufficient

numbers of robots to explore perpetually $\mathcal{COT}$ rings proved in the previous chapter in a fault-free setting (two robots for $\mathcal{COT}$ rings of size three and three robots for $\mathcal{COT}$ rings of size four or more) still hold in the self-stabilizing setting at the price of the loss of anonymity of robots.

The cover time of any exploration algorithm in $\mathcal{COT}$ graphs is unbounded. However, our algorithm using three self-stabilizing robots is speculative: even if its cover time is unbounded in $\mathcal{COT}$ rings, it is bounded in $\mathcal{ST}$ rings. More precisely, its cover time is asymptotically optimal in $\mathcal{ST}$ rings. Therefore, our algorithm solves the perpetual exploration problem in a self-stabilizing way in highly dynamic environments and it solves it in a very efficient way (even when robots may be subject to transient faults) when being executed in static environments.

Our algorithms are the first self-stabilizing algorithms for the problem of exploration, either for static or for dynamic graphs. Moreover our study is the first one to consider the speculative aspect of self-stabilizing algorithms in robot networks.

Chapter 11

Conclusion on Speculation

Contents

Speculative algorithms may be executed in multiple environments and in environments that vary with time. They are used to circumvent high lower bounds (obtained when considering some environments): they solve the problem studied in all the environments considered and they provide good lower bounds to the problem in the most frequent environments in which they will be executed. The goal of this part was to extend the speculative approach to robot networks evolving in dynamic graphs. On this purpose, we have considered the perpetual exploration problem in dynamic rings.

In this part, we first have presented the state of the art about speculative algorithms and about the exploration problem in dynamic graphs. This state of the art has highlighted that (*i*) there is no algorithm using robots that are speculative; (*ii*) there is no algorithm solving the exploration problem while considering robots evolving in $\mathcal{COT}$ graphs; and (*iii*) there is no self-stabilizing algorithm (i.e., algorithm tolerating arbitrary faults such that there exists a time from which these faults no longer occur) solving the exploration problem neither in static graphs nor in dynamic graphs.

In this part we have filled the lack of the state of the art concerning these three points. Indeed, in Chapter 9, we have characterized the necessary and sufficient number of robots permitting to solve the perpetual exploration problem in $\mathcal{COT}$ rings. More precisely, we have proved that two anonymous non-faulty robots are necessary and sufficient to perpetually explore $\mathcal{COT}$ rings of size three, and that three anonymous non-faulty robots are necessary and sufficient to perpetually explore $\mathcal{COT}$ rings of size four or more. Our results on the sufficiency are constructive: we have provided an algorithm solving the perpetual exploration problem in each case. Our general algorithm (the one used in rings of size four or more) is, moreover, a speculative algorithm: it solves the perpetual exploration problem in $\mathcal{COT}$ rings with an unbounded cover time (and it is not possible to do better, refer to Section 8.3), but when being executed in $\mathcal{ST}$ rings, its cover time is bounded and asymptotically optimal.

In Chapter 10, we have proved that the number of non-faulty (anonymous) robots necessary to solve the perpetual exploration problem in $\mathcal{COT}$ rings is equal to the number of self-stabilizing (identified) robots necessary to solve the perpetual exploration problem in $\mathcal{COT}$ rings. We have also proved that these numbers of robots are sufficient by providing two self-stabilizing algorithms. Our algorithm using three self-stabilizing robots and solving the perpetual exploration problem in $\mathcal{COT}$ rings of size four or more is a speculative algorithm. Like the algorithm in the fault-free setting, this algorithm has a bounded and asymptotically optimal cover time when it is executed in $\mathcal{ST}$ rings.

In Chapters 9 and 10, our study was only in $\mathcal{COT}$ rings. However, our analysis can be extended easily to any $\mathcal{COT}$ graph. We present in the section below how to extend our work to general $\mathcal{COT}$ graphs.

11.1 Generalization: Exploration in arbitrary Highly Dynamic Graphs

In this section, we sketch the extension of our work on the characterization on the necessary and sufficient number of robots permitting to solve the perpetual exploration problem in $\mathcal{COT}$ rings to any $\mathcal{COT}$ graph. The sufficiency is based on a generalization of Algorithm $\mathbb{PEF}\_3+$. Our algorithm, solving the perpetual exploration problem in any $\mathcal{COT}$ graph, uses non-faulty anonymous robots. The necessity is based on a generalization of the impossibility proofs given in Section 9.3.1 (which states that two non-faulty anonymous robots cannot perpetually explore $\mathcal{COT}$ rings of size 4 or more) and in Section 9.4 (which states that one non-faulty anonymous robot cannot perpetually explore $\mathcal{COT}$ rings of size 3 or more).

Necessity. In Chapter 9, we have proved that three non-faulty anonymous robots are necessary to perpetually explore any $\mathcal{COT}$ ring (of size four or more). In a $\mathcal{COT}$ ring, there is at most one eventual missing edge.

From this result, we can conjecture that two (non-faulty anonymous) robots per eventual missing edge are needed to "mark" these missing edges, and that one additional robot is necessary to solve the perpetual exploration problem. In other words, we can conjecture that, if there are at most k eventual missing edges in a $\mathcal{COT}$ graph $\mathcal{G}$, then $2k+1$ robots are necessary to solve the perpetual exploration problem in $\mathcal{G}$.

This conjecture is reinforced by the fact that, in the article of Flocchini et al. [86] (refer to Section 8.2.2), if there is no black hole (but there are black edges that make disappear the robots), then the number of robots used by their algorithm to explore the graph is strictly greater than two times the number of black edges.

This intuition can be easily proved by repeating the exact same arguments than the one used in the impossibility proofs of Section 9.3.1 and of Section 9.4 on the following kinds of graphs (also refer to Figure 11.1):

Definition 11.1 (Daisy graph)**.** *A daisy graph G satisfies:*

- *Each cycle of G is simple and contains at least four nodes.*
- *G has at least two cycles.*
- *There is only one node in G (called the central node) having a degree greater or equal to four.*

If a $\mathcal{COT}$ graph $\mathcal{G}$ is based on a daisy graph G composed of k simple cycles, then there are at most k eventual missing edges in $\mathcal{G}$. Indeed, if there were more than k eventual missing edges in $\mathcal{G}$, then the eventual underlying graph of $\mathcal{G}$ would possess at least two distinct connected components and it would not be possible for a node to reach over time any other node infinitely often (which is the property that $\mathcal{COT}$ graphs should respect). To prove that $2k+1$ robots are necessary to solve the perpetual exploration problem in $\mathcal{G}$, we prove that a number of robots less or equal to $2k$ is not able to solve this problem in $\mathcal{G}$. To do so, we initially place the robots in a round robin way on each of the simple cycles such that no robot is on the central node. Note that, initially, there are at most two robots per simple cycle. If there are two robots on a simple cycle, we apply (on this cycle) the same arguments than the one used in Section 9.3.1 that prove that a cycle of four nodes or more cannot be perpetually explored by two robots. This result was proved by showing that at least one node is never visited (because of missing edges that cannot be distinguished from eventual missing edges and force robots to find a different path to explore the cycle). We take back the arguments of this proof in order to prevent the central node to be visited by the robots (refer to Figure 11.2). If there is one robot on a simple cycle, we apply (on this cycle) the same arguments than the one used in Section 9.4 that state that a cycle of three nodes or more

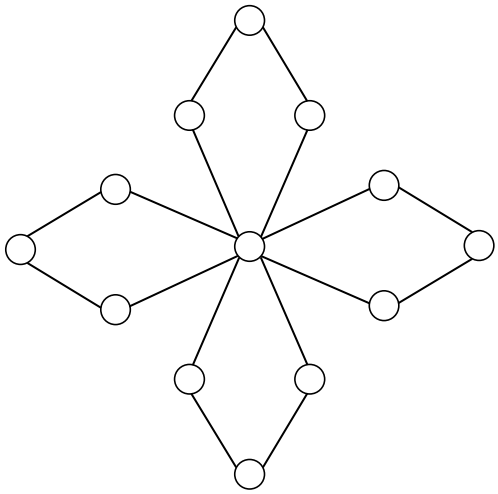


Figure 11.1: Example of a daisy graph possessing 4 cycles of size 4.

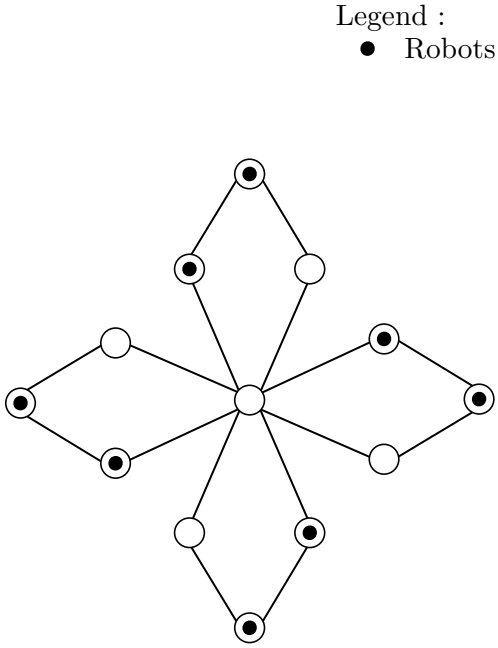


Figure 11.2: Example of the initial placement of 8 robots on a dynamic daisy possessing 4 cycles of size 4.

cannot be perpetually explored by one robot. This result was proved by showing that at least one node is never visited. We take back the arguments of this proof in order to prevent the central node to be visited by the robot during its execution. Hence, none of the robots succeed to visit the central node.

Therefore, at least $2k + 1$ robots are needed to solve the perpetual exploration problem in $\mathcal{COT}$ graphs.

Sufficiency. In this paragraph, we present intuitively an algorithm solving the perpetual exploration problem in any $\mathcal{COT}$ graph $\mathcal{G}$ using a number of robots at least equal to two times the maximal number of possible eventual missing edges in $\mathcal{G}$ plus one. To belong to $\mathcal{COT}$, the eventual underlying graph of a dynamic graph should be connected. Hence, in the worst case, the eventual underlying graph of a $\mathcal{COT}$ graph may be a tree (which possesses $n-1$ edges), implying

that $n-1$ of the edges of its footprint cannot be eventual missing edges. Therefore, the maximal number of eventual missing edges of a dynamic graph is equal to $m-n+1$, where m is the total number of edges of its footprint. Hence, in other words, at least $2*(m-n+1)+1$ robots are used by our algorithm. Note that the robots used by our algorithm do not need to know the number of eventual missing edges of the dynamic graph, they just need to be sufficiently numerous to succeed to solve the perpetual exploration problem. We have studied neither the self-stabilizing aspect nor the speculative aspect of this algorithm.

We consider any undirected $\mathcal{COT}$ graph. We assume that the nodes of the graphs are divided into multiple parts: one for each port of the nodes and one at the middle of the nodes (like this is done in some articles of the state of the art [125, 99]). Moreover, similarly as Di Luna et al. [125] and Gotoh et al. [99], we assume that the robots have access to each port of their current node in mutual exclusion. We assume that the robots are fully-synchronous, anonymous, and do not have the same chirality. We assume that each robot is able to know the exact number of robots located on each part of their current node.

Each robot executes a kind of BFS (Breadth-First Search) [103, 16] which consists in exploring all the adjacent nodes of a current node prior to continuing the exploration in the same way on these adjacent nodes. Each time a robot wants to move in a direction, it moves to the corresponding port in the node. Once a robot is on a port, it stays on this port until the edge connected to this port appears. If this edge never appears, then the robot is stuck and has the role of a sentinel that marks an extremity of an eventual missing edge. If there are at most k eventual missing edges, then at most $2k$ robots are stuck on each extremity of the eventual missing edges. It remains at least one robot that explores infinitely often each of the nodes of the $\mathcal{COT}$ graph: the non-stuck robots execute infinitely often a BFS ignoring the ports that are occupied by robots.

11.2 Perspectives

Short-term perspectives. The algorithms we provided in this part use robots with very few capacities: when the robots are not subject to faults, they are fully-synchronous, anonymous, endowed with weak multiplicity detection and do not have the same chirality; when the robots may be subject to faults, they are fully-synchronous, identified, endowed with strong multiplicity detection and do not have same chirality. Even if the models of robots induced by these computational model and capacities are strong, it could be interesting to study the necessity of these assumptions, and particularly to study the perpetual exploration problem in other models.

The perpetual exploration problem is not comparable to the other forms of exploration (exploration with stop problem, and exploration with return problem), hence doing a similar study as ours for these kinds of exploration is the next step to take. At the opposite, the exclusive perpetual exploration problem is a stronger problem than the perpetual exploration problem, therefore the speculative aspect of this problem in dynamic graphs may be studied to complete our analysis.

In this part, we have focused on the speculative aspect for $\mathcal{ST}$ and $\mathcal{COT}$ graphs. However, the definition of speculation is more general and may be applied to multiple classes of dynamic graphs. Studying the speculative aspect of our algorithms in other classes of dynamic graphs is a short-term perspective.

Finally, in the previous section, we provide an algorithm solving the perpetual exploration problem in $\mathcal{COT}$ graphs, but we did focus neither on the self-stabilizing aspect nor on the speculative aspect of this algorithm. We should analyze this.

Long-term perspectives. In addition to be speculative, one may observe that our algorithms (for non-faulty robots as well as for self-stabilizing robots) are optimal for the cover time we considered (bounded and unbounded) and the classes of dynamic graphs we studied: our algorithms

provide the best cover time (among the one we studied) in $\mathcal{ST}$ and $\mathcal{COT}$ rings. We can generalize this intuition with the following notion:

Definition 11.2 (Optimal speculative algorithm)**.** *Given a model $\mathcal{M}$, a problem $\mathcal{P}$, a set $\mathcal{F} = \{f_0, \ldots, f_k\}$ of functions, with $|\mathcal{F}| \geq 2$, a set $\mathcal{S}_\mathcal{C} = \{\mathcal{C}_0, \ldots, \mathcal{C}_k\}$ of classes of dynamic graphs such that:*

1. *$|\mathcal{S}_\mathcal{C}| \geq 2$.*
2. *For all i, with $0 \leq i \leq k-1, \mathcal{C}_i \subset \mathcal{C}_k$.*
3. *For all i, with $0 \leq i \leq k$, if $\mathcal{C}_i \subset \mathcal{C}_j$ then $f_i|_{\mathcal{C}_i} \in O(f_j|_{\mathcal{C}_j})$.*
4. *There exists at least one i, with $0 \leq i \leq k-1$, such that $\mathcal{P}$ is not solvable in $\Theta(f_i|_{\mathcal{C}_i})$ in $\mathcal{C}_k$ (implying that $f_i|_{\mathcal{C}_i} \in o(f_k|_{\mathcal{C}_k})$).*

a $(\mathcal{P}, \mathcal{F}, \mathcal{S}_\mathcal{C})$-optimal speculative algorithm $\mathcal{A}$ for $\mathcal{P}$ under $\mathcal{M}$ is such that:

- *$\mathcal{A}$ is a $(\mathcal{P}, \mathcal{F}, \mathcal{S}_\mathcal{C})$-speculative algorithm for $\mathcal{P}$ under $\mathcal{M}$.*
- *For all i, with $0 \leq i \leq k$, if $\mathcal{C}_i \subset \mathcal{C}_j$ and $\mathcal{A}$ does not solve $\mathcal{P}$ under $\mathcal{M}$ in $\Theta(f_i|_{\mathcal{C}_i})$ in $\mathcal{C}_j$ this is because $\mathcal{P}$ is not solvable in $\Theta(f_i|_{\mathcal{C}_i})$ in $\mathcal{C}_j$ graphs under $\mathcal{M}$.*

Our algorithms are then optimal speculative perpetual exploration algorithms with respect to $\mathcal{ST}$ and $\mathcal{COT}$. A long-term perspective is to analyze if our algorithms are optimal speculative algorithms (for non-faulty and self-stabilizing robots) with respect to classes of dynamic graphs other than $\mathcal{ST}$ and $\mathcal{COT}$.

Similarly, for all the problems that are solvable in $\mathcal{COT}$ graphs, and that admit speculative solutions, it could be interesting to find solutions that are optimal speculative algorithms.

The previous definition does not exclude the possibility for an algorithm to be an optimal speculative algorithm solving the problem studied with a measurement of performance different from the best possible. Therefore, we can define an optimum notion for a speculative algorithm: an optimum speculative algorithm is a speculative algorithm that solves the problem studied with the best measurement of performance possible in each of the classes of dynamic graphs considered. Such an algorithm is necessarily an optimal speculative algorithm.

Our algorithms are optimum speculative algorithms with respect to $\mathcal{ST}$ and $\mathcal{COT}$. Indeed, our algorithms solve the perpetual exploration problem with an unbounded cover time while being executed in $\mathcal{COT}$ rings (and it is not possible to do better, refer to Section 8.3) and they solve the perpetual exploration problem in an asymptotically optimal cover time in $\mathcal{ST}$ rings.

The same perspectives as previously are possible to explore: study if our algorithms are optimum speculative algorithms in classes of dynamic graphs other than $\mathcal{ST}$ and $\mathcal{COT}$; and find optimum speculative algorithms for each of the problems solvable in $\mathcal{COT}$ graphs that admit speculative solutions.

Part IV

Conclusion of the Thesis

Chapter 12

Concluding Remarks

Contents

12.1 Sum Up of the Main Parts

In this thesis, we considered distributed systems, i.e., systems composed of multiple processes, each executing algorithms (ordered sequence of instructions) in an autonomous manner. Algorithms for distributed systems are hard to conceive since all the entities have to collaborate to solve a same task and each entity has to take decisions based its own vision of the system. However, distributed systems are convenient: thanks to the multiple entities composing it, they are more tolerant to faults, it is possible to have some entities that are faulty and still be able to solve the wanted task. Moreover processes of a distributed system can divide between them parts of a same task making its execution quicker than if it was performed by a single process.

Generally, the goal of the study of distributed systems is to determine the models (set of assumptions made on the distributed system) in which the problem considered is solvable and the ones in which it is impossible to solve; and find an algorithm for the models in which it is solvable. In particular, the classical approach is to conceive an algorithm for a specific model. There is no guarantee on a correct behavior of the algorithm in the case where it is executed in a model other than the one for which it was designed.

In this thesis, we were interested in the study of algorithms that are conceived to be executed in multiple different models (i.e., they are not conceived only for a specific model). These algorithms adapt to the model in which they are executed. We present below some of these approaches that we considered in this thesis.

Indulgent algorithms [5] are conceived to be executed even in models in which the problem studied is impossible to be solved: they guarantee correctness when the system is synchronous, and when the system is asynchronous and the problem studied becomes impossible then they maintain the safety of the problem. Hence, when the system is asynchronous, making the problem impossible, only a degraded version of the problem is considered. Therefore, indulgent algorithms are convenient when considering problems impossible in asynchronous systems and when the synchronicity of the environment in which they will be executed is not known in advance, or when the synchronicity of the environment changes with time.

The indulgent approach only considers that the synchronicity of the system may be unknown in advance or change with time. However, many parameters may be unknown in advance or change with time. In particular, it is possible to represent some distributed systems thanks to dynamic graphs where the nodes represent the processes and the edges represent the possibility for two processes to communicate, and in which the nodes and edges appear and disappear with time. There exist, depending on the frequencies of appearance and disappearance of the nodes and edges, different kinds of dynamic graphs. There exists a classification that defines different kinds of dynamic graphs (in which only the edges may appear and disappear with time) [40, 38]. As a designer of algorithms, it is not always possible to know in which kind of dynamic graphs our algorithms are used.

The gracefully degrading approach [20] is based on the same principle as the indulgent approach except that, instead of the synchronicity, this is the dynamics of the system that is considered to be unknown in advance or to change with time. In other words, gracefully degrading algorithms guarantee correctness when they are executed in a dynamic system in which the problem $\mathcal{P}$ studied is solvable but they satisfy weaker specifications than $\mathcal{P}$ that ensure its safety otherwise. Therefore, gracefully degrading algorithms are useful when considering problems impossible in some kinds of dynamic graphs and when the dynamics of the system in which they will be executed are not known in advance, or when the dynamics of the system change with time.

Moreover, generally, while studying distributed algorithms, only the worst case is analyzed: an upper bound (in the worst case) to solve the problem studied is computed. However, the worst case is not always the most frequent to happen. There exist in the literature speculative algorithms [114, 77, 9] which are algorithms optimized for the most frequent case: they satisfy the specification of the problem studied in any execution in which they are susceptible to be executed but also they are very efficient for the executions that are more likely to happen. The efficiency of a speculative algorithm is analyzed, depending on the context, thanks to a complexity in time, in memory, ... Note that even if speculative algorithms focus on optimizing the most frequent executions, it is an optimization of the worst case in order to provide an upper bound for all the most frequent executions. Speculative algorithms circumvent high lower bounds: the upper bound reached by a speculative algorithm to solve a problem in the most frequent environments is lower than the lower bound required to solve the problem considering all the possible environments. Such algorithms are useful when the environment in which they will be executed is not known in advance and when it is possible to optimize the solutions of the problem studied for some of the executions.

Graceful degradation and speculation are two orthogonal notions: the former permits to circumvent impossibility results, while the latter focus on optimizing results in frequent executions. Generally, algorithms consider only the environments in which the problems studied are solvable saying that the problematic environments (in which the problems are impossible) are not frequent. Therefore, we can say that the graceful degradation focus on not frequent executions while speculation focus on frequent one. Refer to Figure 12.1 to see an overview of these two notions.

In this thesis, we studied distributed systems made of robots: the processes composing the distributed system are robots (autonomous entities able to move that are endowed with sensors to sense their environment). The robots we considered are evolving in dynamic graphs in which only the edges may appear and disappear with time and where the nodes represent the locations where the robots may be and the edges represent the possibility for a robot to move from one location to another one. There is no article studying the speculative or gracefully degrading aspects of algorithms using robots. The goal of this thesis was to determine if it was possible to apply these two approaches to robot networks, and more particularly to robots evolving in dynamic graphs. We answered positively to this interrogation: the main contribution of this thesis being the extension of the gracefully degrading and speculative notions to robot networks evolving in dynamic graphs.

More precisely, in Chapter 6, we extended the gracefully degrading notion to robot networks evolving in dynamic graphs. In this chapter, we studied the gathering problem (in which all robots terminate their execution on the same node of the graph in finite and bounded time) in dynamic rings. This problem is a good case study, since it is impossible to solve in $\mathcal{AC}$ (graphs connected at each instant time) rings [122]. We provided a gracefully degrading gathering algorithm. This algorithm solves the specification of the gathering problem in $\mathcal{ST}$ (in which each edge is present at each instant time) and $\mathcal{BRE}$ (in which each edge is present at least once every δ units of time) rings. However, in $\mathcal{RE}$ (in which each edge of the graph is infinitely often present) rings, since it is not possible to solve the original specification of the gathering problem, it solves a weaker version of the gathering problem in which all robots terminate their execution on the same node

Legend:

$\mathcal{C}_x$ Classes of dynamic graphs

Any execution of $\mathcal{A}$ solves $\mathcal{P}$

Any execution of $\mathcal{A}$ solves at least a problem weaker than $\mathcal{P}$

Any execution of $\mathcal{A}$ solves $\mathcal{P}$ more efficiently
(for instance in terms of time complexity)

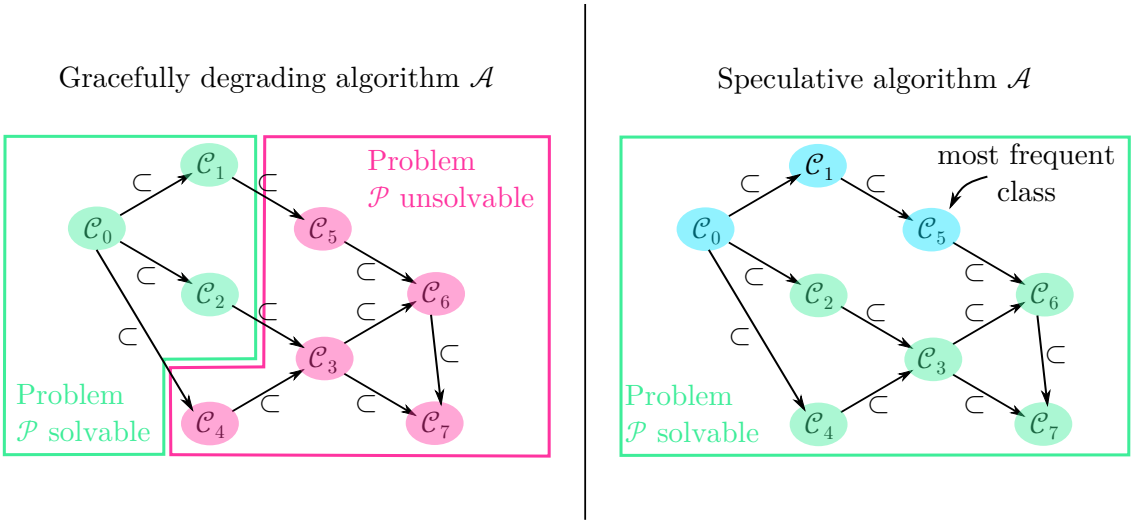


Figure 12.1: Graceful degradation and speculation.

of the graph in finite (but not necessarily bounded) time. Similarly, as mentioned previously, the specification of the gathering problem is not solvable in $\mathcal{AC}$ rings. Our algorithm solves the weak gathering problem in which all robots but (at most) one terminate their execution on the same node of the graph in finite and bounded time in $\mathcal{AC}$ rings. Moreover, the gathering problem is also impossible in $\mathcal{COT}$ (in which there exists infinitely often a path over time between any pair of nodes) rings, since $\mathcal{AC} \subset \mathcal{COT}$. Our algorithm solves the eventual weak gathering problem in which all robots but (at most) one terminate their execution on the same node of the graph in finite (but not necessarily bounded) time in $\mathcal{COT}$ rings (the weak gathering problem being also impossible to solve in $\mathcal{COT}$ rings).

In Chapter 9 and in Chapter 10, we provided speculative algorithms for robot networks. We dealt with the perpetual exploration problem (in which all the nodes of the graph have to be visited infinitely often by at least one robot). In each of these chapters, we have characterized the necessary and sufficient number of robots needed to perpetually explore $\mathcal{COT}$ rings. In Chapter 9, this analysis has been performed on non-faulty robots while in Chapter 10 it has been done on self-stabilizing robots (robots subject to transient faults, i.e., arbitrary faults such that there exists a time from which these faults no longer exist). In both cases, we have proved that two robots are necessary and sufficient to solve the perpetual exploration problem in $\mathcal{COT}$ rings of size 3; and that three robots are necessary and sufficient to solve the perpetual exploration problem in $\mathcal{COT}$ rings of size four or more. Our results for the sufficiency are constructive: we proposed algorithms to show the sufficiency. The algorithms provided while considering rings of size four or more (in the non-faulty case and in the self-stabilizing case) are speculative. They are speculative considering the cover time which is the worst time of the minimal time taken by the robots to

explore at least once each node of the ring from any time in all the possible executions of their algorithm. Indeed, they solve the perpetual exploration problem in $\mathcal{COT}$ rings in an unbounded cover time, and it is not possible to do better, and they solve this problem in an asymptotically optimal cover time in $\mathcal{ST}$ rings (i.e., $\Theta(n - \mathcal{R})$, with n the size of the ring and $\mathcal{R}$ the number of robots in the system, for the non-faulty case, and $\Theta(n)$ for the self-stabilizing case).

Besides these results, we present, in Chapter A, a work solving the approach problem in the plane (i.e., two robots, endowed with a limited visibility range, have to be located, in finite time, in the range of vision of each other). Our algorithm uses asynchronous robots endowed with weak orientation capabilities. It solves the problem in a polynomial number of moves.

12.2 Perspectives of the Thesis

There exist multiple future work related to this thesis. We have already presented the perspectives about our work on the graceful degradation in Chapter 7 and the perspectives about our work on the speculation in Section 11.2. In this section, we present more general perspectives.

First of all, one may observe that we have only considered that robots may be subject to transient faults. Hence, we can study speculative and gracefully degrading approaches while using robots subject to other faults like Byzantine faults (which are arbitrary faults). Contrary to a transient fault, a Byzantine fault can occur at any time in the execution. Byzantine faults include transient faults and are therefore more difficult to handle. Handling Byzantine faults permit to have very robust algorithms, they tolerate any fault (i.e., they satisfy the specification of the problem considered whatever the faults). Generally, to deal with Byzantine faults the ratio between the number of non-faulty robots and Byzantine robots is important: to tolerate Byzantine faults, all the algorithms of the state of the art assume that at least the majority of the robots are non-faulty [33, 68, 25]. The first step to study Byzantine robots, in the context of this thesis, would be to characterize the necessary and sufficient number of non-faulty robots needed to solve problems while considering a gracefully degrading or speculative approach in dynamic graphs.

Moreover, since it is possible for the edges of $\mathcal{COT}$ rings to be missing initially and this for an arbitrary long time, when considering the speculation, we proved that the cover time of any algorithm solving the perpetual exploration problem in $\mathcal{COT}$ rings is unbounded (refer to Section 8.3). If we had been interested in the complexity in terms of movements (i.e., the worst minimal number of edges traversed by the robots to visit each node of the ring in all the possible executions), the complexity would have been also unbounded. Indeed, the robots have to perpetually explore an unknown dynamic ring, hence if there exists during a while two distinct connected components of this ring, with all the robots only in one of these connected components, the robots will explore it infinitely often, making increase the number of movements of the robots without exploring at least once each of the nodes of the ring. Indeed, the robots have no knowledge about the graph, therefore they will explore infinitely often the nodes of the connected component in which they are located, without succeeding to solve the perpetual exploration of the whole ring. Hence, these unbounded complexities are due to the dynamics of the environment, not to the algorithms (that are forced by the dynamics to act the way they act). It is not fair to compare complexities in $\mathcal{ST}$ graphs and in $\mathcal{COT}$ graphs. Indeed, a robot in a static graph that wants to cross an edge is always able to do so, whereas a robot in a dynamic graph that wants to cross an edge is not always able to do so since this edge could be missing. In this last case, the robot could either wait for the missing edge to appear (making then the time complexity increase) or find another path to reach its destination (making the time complexity as well as the complexity in terms of movements increase). Hence, it is important to introduce a new notion of time complexity and a new notion of complexity in terms of movements for dynamic graphs. This work would permit to compare the efficiency of algorithms for dynamic graphs (whatever the dynamics considered), like it has been done in the message passing model to

compare synchronous and asynchronous algorithms in introducing the notion of rounds [8]. Note that this notion of rounds, even if introduced for the message passing model, is also suitable to compare synchronous and asynchronous algorithms using robots. Dubois et al. [78] already tried to define a new notion of time complexity for dynamic graphs adapted for the message passing model. In their definition, the time unit (i.e., the notion of round) of an execution is the longest time between the sending of a message through a non-eventual missing edge and its reception. However, contrary to messages, robots cannot be duplicated or broadcast, and they sometimes must change direction while stuck on a missing edge. The notion of time complexity of Dubois et al. would count the time wasted by the robots while changing direction, but, we do not want to take it into account. Hence, even if this time complexity could be suitable for the message passing model it is not suitable for robot networks. Therefore, a challenging future work is to define a time complexity suitable for any dynamic distributed system; and define a complexity in terms of movements suitable for robots evolving in dynamic graphs.

Once these complexities defined, it would be worth it to analyze again our algorithms, and see if they are still speculative.

Besides, we have analyzed speculative and gracefully degrading aspects separately. At first sight, it is not possible to combine the two notions since a gracefully degrading algorithm considers a problem impossible in some environments, and since a speculative algorithm considers a problem solvable in any execution in which it is susceptible to be executed. However, it could be interesting to consider the speculative aspect of gracefully degrading algorithms: for each set of classes in which a gracefully degrading algorithm solves a problem $\mathcal{P}_i$, optimize this problem for the class, among this set, that is the most frequent. Of course, this implies that the problem $\mathcal{P}_i$ should be solvable in multiple classes of dynamic graphs to analyze the speculative aspect of the gracefully degrading algorithm on this set of classes. For instance, in our study, our gracefully degrading gathering algorithm solves the specification of the gathering problem in $\mathcal{BRE}$ and $\mathcal{ST}$ rings, therefore, it is possible to study the speculative aspect of this algorithm for these two classes. Similarly, if our algorithm had solved the eventual gathering problem in another class of dynamic graphs than $\mathcal{RE}$ graphs, it would have been possible to study the speculative aspect of our algorithm for these two classes.

Part V

Appendix

Appendix A

Approach in the Plane or Rendezvous in an Infinite Grid

Contents

In this chapter, we present another problem that we studied, but that is not entirely in the scope of this thesis. The problem studied is the deterministic approach in the plane. The approach problem consists for two robots evolving in the plane, endowed with a limited visibility range, to be located, in finite time, in the range of vision of each other. Generally, while considering the approach problem, each robot is able to see its environment in a range of vision equals to one unit of length.

We provide a deterministic algorithm that uses asynchronous robots and that solves the approach problem in a number of movements polynomial in the initial distance separating the two robots and in the size of the binary representation of the smallest identifier. This work has been published in DISC 2017 [22], in Distributed Computing [24] and in ALGOTEL 2018 [23].

Before presenting succinctly our work, we first present a brief overview of the state of the art about this subject.

A.1 State of the Art about the Approach in the Plane

As indicated in Section 3.1.4, the approach in the plane problem can be reduced to the rendezvous problem in an infinite grid (i.e., two initially scattered robots have to be located, in finite time, on the same node of the infinite grid or on the same point of an edge of the grid) [51, 67].

Indeed, consider an infinite square grid with edge length 1. More precisely, for any point v in the plane, we define the grid G_v as the infinite graph such that v is a node of G_v. Every node u of G_v is adjacent to 4 nodes at Euclidean distance 1 from it, and located north, east, south, and west from node u. We now focus on how to transform any rendezvous algorithm in the grid G_v to an algorithm for the task of approach in the plane (refer to Figure A.1 for an illustration of the explanation given below).

Let $\mathcal{A}$ be any rendezvous algorithm for any basic infinite grid. Algorithm $\mathcal{A}$ can be executed in the grid G_v, for any point v in the plane. Consider two robots in the plane starting respectively from point v and point w (of the plane). Let V' be the set of nodes in G_v that are the closest nodes from w. Let v' be a node in V', arbitrarily chosen. Note that v' is at distance at most $\sqrt{2}/2 < 1$ from w. Let α be the vector $v'w$. Execute Algorithm $\mathcal{A}$ on the grid G_v with the two robots starting respectively at nodes v and v'. Let p be the point in G_v at which the two robots meet at some time t. Note that p is either a node or a point inside an edge of G_v. The transformed algorithm $\mathcal{A}^*$ solving the approach in the plane works as follows: execute the same algorithm $\mathcal{A}$ but with one robot starting at v and traveling in G_v and the other robot starting at w and traveling in G_w, so that the starting time of the robot starting at w is the same as the starting time of the robot starting at v' in the execution of $\mathcal{A}$ in G_v. The starting time of the robot starting at v does not change. If the approach has not been accomplished before, at time

t, the robot starting at v and traveling in G_v is at point p (as this was the case while executing $\mathcal{A}$). In the same way, the robot starting at w and traveling in G_w reaches some point q at time t such that $q = p + \alpha$. Hence, at time t, the two robots are at distance less than one from each other, which means that they accomplish the approach in the plane (since the range of vision of each robot is equal to one unit of length).

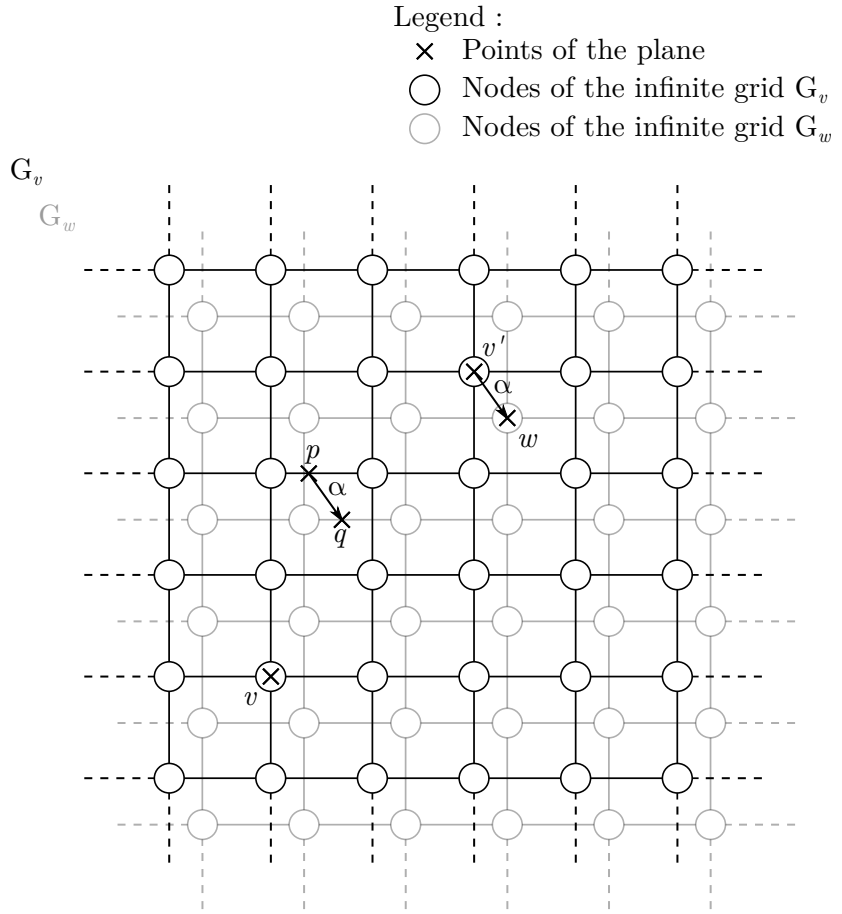


Figure A.1: Transformation of any rendezvous algorithm in the grid G_v to an algorithm for the task of approach in the plane.

The cost (in terms of the number of movements executed by each robot) of the rendezvous algorithm $\mathcal{A}$ is either greater than or equal to the cost of the algorithm solving the approach problem in the plane thanks to $\mathcal{A}^*$.

Our goal is to study the approach problem in a harsh setting. More precisely, we want to focus on the deterministic approach problem executed by asynchronous robots endowed with very few capacities. On this purpose, we present here the four articles the more related to our work. All the work presented below consider the deterministic approach problem and study the rendezvous problem in an infinite grid to solve it. Note that each of these articles uses robots having access to distinct information like distinct identifiers. This is mandatory to solve the deterministic approach problem; otherwise the robots could execute the same actions at the same instant time and hence be separated from the same distance at each instant from time $t = 0$, making the deterministic approach problem impossible.

Czyzowicz et al. [51] consider robots that evolve in any (finite or infinite) anonymous graph. The robots they consider are asynchronous. Each robot possesses a distinct positive integer as identifier. The robots know only their own identifier. They have persistent memory, and no restriction is made on the memory of the robots. They do not know any information on the graph

nor on the initial distance separating them. Even if their algorithm solves the rendezvous problem in a strong model (asynchronism of the robots, no knowledge of a common global coordinate system for the robots), its cost (in terms of the number of edge traversals) to solve the problem is exponential in the size of the graph.

Collins et al. [47] provide an algorithm that solves the rendezvous problem in an infinite grid with a number of edge traversals polynomial in the initial distance separating the two robots. More precisely, the complexity of their algorithm is in $O(d^{2+\epsilon})$ for any constant $\epsilon > 0$, where d is the initial distance between the robots. The robots they study are asynchronous. However, the robots are assumed to have access to a common global coordinate system. Each robot knows its initial position (i.e., its coordinates in this coordinate system) on the grid, but does not know the initial position of the other robot. Therefore, even if the complexity of their algorithm is interesting (since the lower bound on the number of edge traversals to solve the rendezvous problem is in $\Omega(d^2)$), the model they consider is unfortunately a weak one.

With the exact same assumptions on robots than those made by Collins et al. [47], Bampas et al. [13], present an algorithm where the robots evolve in multidimensional infinite grids. Their algorithm achieves the rendezvous in a number of edge traversals in $O(d^{\delta} * log(d)^{\epsilon})$ for some $\epsilon > 0$, with δ the dimension of the grid and where d is the initial distance separating the two robots. Therefore, their algorithm has a better complexity than the one proposed by Collins et al. [47]. In order to suppress the knowledge about the initial position on the common global coordinate system, in the same article, the authors propose a method to label the ports of the nodes. This labeling of the ports permits the robots to extract the label of the node where they are located and permits them to have enough information to solve the rendezvous problem using the same algorithm as the one performed when they are aware of their initial position.

The work that is the most related to ours is the one of Dieudonné and Pelc [67]. In this article, the authors consider robots evolving in an infinite grid. They possess distinct identifiers, a compass (but they do not have access to a common global coordinate system), and the same unit of length. Their algorithm solves the rendezvous problem with a number of edge traversals polynomial in the initial distance separating the two robots and in the size of the binary representation of the smallest identifier. However, the robots are not fully asynchronous. Indeed, they may possess distinct speeds, but all along the execution of the algorithm, a robot possesses always the same speed (i.e., the speed cannot evolve with time). Moreover, even if a robot does not know the speed of the other robot, it is aware of its speed.

A.2 Our Approach in the Plane

The asynchronous model is the more realistic model. As shown in the previous section, none of the articles of the state of the art solve the approach in the plane problem in a number of edge traversals polynomial while robots are asynchronous, or if they do so this is at the price of important assumptions (such as robots possess the same coordinate system [47, 13], or the ports of the nodes are labeled such that the nodes of the infinite grid are identified [13], or the robots are not moving in a fully asynchronous manner [67]).

We therefore decide to fill the lack of the state of the art by providing a solution to the approach in the plane problem at polynomial cost (in terms of the number of edge traversals) while considering fully asynchronous robots endowed with very few capacities.

More precisely we made the following assumptions on the robots. The asynchronous robots are endowed with compasses (i.e., robots agree on the direction and orientation of the two cardinal axes). Robots also agree on a same unit of length. However, they do not have a common coordinate system, in particular, they do not know an origin point of the coordinate system. The robots have identifiers, but each robot knows only its own identifier. The robots possess the same range of vision. The two robots are initially scattered, and do not know the initial distance that separates them. Moreover, the robots do not start necessarily the execution of their algorithm at

the same time. They are indeed woken at arbitrary time.

We describe in the following the intuition of our algorithm solving the approach problem. Like the other algorithms of the state of the art, we study the rendezvous in an infinite grid in order to solve the approach problem. The grid we consider is anonymous, i.e., there is no identifier on the nodes.

Once woken, each robot r first modifies its identifier id_r: each bit of the binary representation of the identifier is duplicated and the bits "01" are added at the end of the sequence of these duplicated bits. More formally, the binary representation of id_r, denoted $b_1 b_2 \dots b_{|id_r|}$, is transformed in $b_1 b_1 b_2 b_2 \dots b_{|id_r|} b_{|id_r|} 01$. Call $transformedIdentifier_r$ the transformed identifier of the robot r, and let $transformedIdentifier[i]$, for any $1 \leq i \leq |transformedIdentifier_r|$, be its i^{th} bit.

The two robots r_1 and r_2 possess distinct identifiers. Therefore, there exists a position i (with $1 \leq i \leq min\{|transformedIdentifier_{r_1}|, |transformedIdentifier_{r_2}|\}$) such that $transformedIdentifier_{r_1}[i] \neq transformedIdentifier_{r_2}[i]$.

The goal of the rendezvous algorithm is for the two robots to meet on a node or an edge of the infinite grid. To do so, the two robots read the bits of their transformed identifier and execute some patterns depending on the values of these bits. More precisely, when a robot reads a bit equal to zero, it executes a certain pattern $\mathcal{P}_0$ starting on its initial node. The pattern $\mathcal{P}_0$ ends on the node where the robot started performing this pattern. When a robot reads a bit equal to one, it performs a pattern $\mathcal{P}_1$ starting on its initial node, such that $\mathcal{P}_1$ corresponds to the execution of the pattern $\mathcal{P}_0$ on each node at a certain distance d from its current node. Like the pattern $\mathcal{P}_0$, $\mathcal{P}_1$ ends on the node where it was initiated. The pattern $\mathcal{P}_1$ encompasses the pattern $\mathcal{P}_0$. Therefore, if r_1 executes the pattern $\mathcal{P}_0$ (of $\mathcal{P}_1$) initiated on the initial node of the robot r_2 (note that this means that r_1 considers a distance d greater than or equal to the initial distance between r_1 and r_2) while r_2 executes $\mathcal{P}_0$, the rendezvous will occur. However, as the pattern $\mathcal{P}_1$ corresponds to multiple patterns $\mathcal{P}_0$, it is possible that r_2 finishes to execute its pattern $\mathcal{P}_0$ before the robot r_1 is able to execute the pattern $\mathcal{P}_0$ (of $\mathcal{P}_1$) initiated on the initial node of r_2. This is why we introduce a circular shift: each bit of the transformed identifier is processed s times, where s corresponds to the number of nodes at distance d from the initial node of the robot. For each new value of s the first pattern $\mathcal{P}_0$ of $\mathcal{P}_1$ is initiated on a new node among the s possible one.

As indicated, the rendezvous occurs if the distance d considered is greater than or equal to the initial distance between the two robots. However, the two robots do not know the initial distance that separates them. To circumvent this problem, the robots make assumptions on this distance. First, the two robots assume that the initial distance separating them corresponds to one unit of length. For each distance assumed, the robots execute the rendezvous algorithm, and if the rendezvous has not occurred with the current assumption, they make a new assumption: they assume that the initial distance separating them corresponds to twice the previous one.

For each distance d assumed, the robots read d bits of their transformed identifier. In the case where the length of the transformed identifier of a robot r is smaller than d, once the robot has read its full transformed identifier, we assume that it reads only bits equal to zero.

For the rendezvous to occur, we want the two robots to perform simultaneously two distinct bits at the same position of their transformed identifiers and with the same assumption on d.

Thus, despite the fact that the robots are asynchronous, we want them to be synchronized in their execution. To do so, two kinds of synchronization mechanisms are used: a synchronization on the distance (i.e., the two robots make the same assumption on the distance at the same time) and a synchronization on the bits read (i.e., for all i, with $1 \leq i \leq min\{|transformedIdentifier_{r_1}|, |transformedIdentifier_{r_2}|\}$, the two robots must read the i^{th} bit of their respective transformed identifiers at the same time).

The goal of these synchronization mechanisms is to make the slower robot catch up its delay compared to the other robot. A robot that executes these synchronization mechanisms performs patterns that force the other robot to increase its speed; otherwise the rendezvous occurs. Each time a robot finishes to read a bit, or each time it wants to make a new assumption on the

distance, it executes one of these synchronization mechanisms.

To sum up, thanks to these synchronization mechanisms, we know that there exists a time at which the two robots perform simultaneously two distinct bits (since there exists an integer i, $1 \leq i \leq min\{|transformedIdentifier_{r_1}|, |transformedIdentifier_{r_2}|\}$, such that $transformedIdentifier_{r_1}[i] \neq transformedIdentifier_{r_2}[i]$) while considering a distance d greater than or equal to the initial distance that separates them. Therefore, at this time, the rendezvous occurs. Hence, our algorithm succeeds to solve the approach in the plane problem using asynchronous robots possessing few capacities. Moreover, the patterns are carefully chosen in order that the number of edge traversals is polynomial. More precisely, our algorithm solves the approach problem in a number of edge traversals polynomial in the initial distance between the two robots and in the size of the binary representation of the smallest identifier.

Appendix B

Résumé français de la thèse

Contents

Dans cette thèse, nous étudions des systèmes composés de plusieurs processus capables de communiquer ensemble. De tels systèmes sont dits distribués. Chaque processus d'un système distribué exécute un algorithme (local) (i.e., une séquence ordonnée d'instructions) d'une manière non coordonnée, i.e., sans l'aide d'aucune entité centrale. Un algorithme distribué est l'ensemble de tous les algorithmes locaux des entités d'un système distribué.

Les systèmes distribués sont à opposer aux systèmes centralisés dans lesquels une seule entité a une vue globale du système et prend toutes les décisions. Dans les systèmes distribués, tous les processus prennent leurs propres décisions en fonction de leurs propres connaissances du système. Ils ont leur propre horloge et leur propre vitesse de calcul. Afin de résoudre des problèmes, les processus doivent coopérer tous ensemble bien qu'ils n'aient ni la même perception du temps, ni la même vision du système...

Même si leur conception peut être délicate à divers égards, en particulier la synchronisation des algorithmes locaux, les systèmes distribués présentent certains avantages. En effet, les processus peuvent s'exécuter en parallèle, ce qui rend les exécutions de tâches plus rapides que si elles étaient effectuées par un seul processus. De plus, puisqu'il y a plusieurs processus pour exécuter une tâche, il est possible de tolérer des fautes: si l'un des processus arrête d'exécuter correctement son algorithme il peut encore être possible de résoudre le problème.

Un système distribué est un modèle général qui permet de représenter différents types de systèmes réels tels que les réseaux téléphoniques, les réseaux de transport, Internet... Il existe plusieurs hypothèses qui peuvent être faites sur un système distribué pour représenter tous ces divers systèmes réels : les entités sont-elles synchrones (leur vitesse de calcul est bornée) ou sont-elles asynchrones (leur vitesse de calcul est finie mais non bornée) ? Les entités sont-elles capables de communiquer ? Comment communiquent-elles (grâce à une mémoire partagée ou à des envois de messages) ? Les entités sont-elles capables de communiquer de manière synchrone (le temps d'acheminement des messages est borné) ou asynchrone (le temps d'acheminement des messages est fini mais non borné)? Les entités peuvent-elles être sujettes à des fautes ? Est-ce que le graphe de communication des entités est représenté par un graphe statique ou par un graphe dynamique (où les arêtes modélisent la possibilité pour deux entités de communiquer, et peuvent apparaître et disparaître avec le temps) ?

Un environnement correspond à l'ensemble des hypothèses faites sur un système distribué. Certains environnements sont plus difficiles que d'autres : par exemple un environnement composé d'entités asynchrones est plus difficile que le même environnement où les entités sont synchrones. Quand un environnement est trop dur, certains problèmes deviennent impossibles à résoudre ou les bornes inférieures (i.e., la performance minimale selon certaines métriques comme le temps utilisé, la mémoire utilisée, le nombre de messages envoyés, *etc.*, nécessaire pour résoudre un problème) pour résoudre certains problèmes augmentent. Par exemple, le problème du consensus (où les processus doivent décider de manière irrévocable, en temps fini, la même valeur

parmi un ensemble de valeurs initialement proposées par chacun des processus) est impossible lorsque les processus sont asynchrones et peuvent arrêter l'exécution de leur algorithme [82].

L'approche classique, lors de l'étude d'algorithmes distribués, consiste à analyser la faisabilité des problèmes, i.e., déterminer les environnements dans lesquels les problèmes étudiés sont résolubles et ceux dans lesquels ils sont impossibles à résoudre. Une fois que ces environnements ont été déterminés, l'approche classique consiste à trouver au moins un algorithme par environnement où les problèmes peuvent être résolus et de préférence avec de bonnes performances (comme une bonne complexité temps). Un algorithme conçu pour un environnement particulier ne garantit ni un comportement correct ni de bonnes performances lorsqu'il est exécuté dans un autre environnement. Cette approche peut être restrictive, par exemple dans un contexte où nous ne pouvons pas savoir à l'avance l'environnement dans lequel l'algorithme sera réellement exécuté. Ceci est particulièrement vrai lorsque l'algorithme est exécuté dans un environnement qui peut changer avec le temps, comme par exemple un environnement où les processus peuvent être synchrones à un moment donné et asynchrones à un autre moment ou un environnement qui est plus ou moins dynamique (i.e., la fréquence d'apparition et de disparition des arêtes du graphe de communication des processus est plus ou moins élevée) en fonction du temps.

Dans cette thèse, nous adoptons une approche alternative en étudiant des algorithmes qui s'auto-adaptent à l'environnement dans lequel ils sont exécutés. Nous nous concentrons sur deux de ces approches: l'*indulgence* / la *dégradation progressive* et la *spéculation* que nous décrivons ci-dessous.

Algorithmes indulgents/progressivement dégradants. Les algorithmes indulgents [5, 118, 131] et les algorithmes progressivement dégradants [20] (que nous définissons par la suite) sont basés sur le même principe mais appliqués à différents environnements : les algorithmes indulgents se concentrent sur des environnements où seule la synchronicité change, alors que les algorithmes progressivement dégradants se concentrent sur des environnements où seule la dynamicité du système change. Lorsque de tels algorithmes sont exécutés dans un environnement où le problème étudié est impossible, ils ne peuvent évidemment pas résoudre le problème, mais ils garantissent de faire de leur mieux. En effet, ils garantissent que tant que les conditions du système ne conviennent pas pour résoudre le problème, ils résolveront une version dégradée de ce dernier.

Par exemple, Biely et al. [20] ont présenté un algorithme progressivement dégradant qui résout le problème du consensus dans les systèmes où les hypothèses de connectivité du graphe de communication sont fortes (les arêtes sont fréquemment présentes), et qui se dégradent progressivement en k-accord (où les processus doivent décider irrévocablement, en temps fini, k valeurs parmi un ensemble de valeurs initialement proposées par chacun des processus) lorsque la connectivité entre les processus communiquants diminue et rend le problème du consensus impossible.

Plus précisément, les algorithmes indulgents / progressivement dégradants résolvent le problème $\mathcal{P}$ étudié quand ils sont exécutés dans certains environnements où ce problème est résoluble et résolvent des problèmes plus faibles que $\mathcal{P}$ dans les autres cas. En calcul distribué, les définitions des problèmes sont généralement décomposées en une propriété de sûreté et une propriété de vivacité. Intuitivement, selon Lamport [117] "une propriété de sûreté est une propriété qui dit que quelque chose ne va pas arriver" et "une propriété de vivacité est une propriété qui stipule que quelque chose doit se produire". Les problèmes plus faibles que $\mathcal{P}$, résolus par un algorithme indulgent / progressivement dégradant, peuvent conserver la sûreté de $\mathcal{P}$ pour préserver l'essence de ce problème.

Ces approches sont un moyen de contourner les résultats d'impossibilité qui se produisent dans certains environnements car les algorithmes indulgents / progressivement dégradants fournissent une solution de type "au mieux" quand ils sont exécutés dans un environnement où le problème est impossible. De tels algorithmes sont également utiles lorsque l'environnement dans lequel ils sont exécutés n'est pas connu à l'avance, ou lorsque l'environnement peut évoluer avec

le temps car ils s'adapteront à ces changements sans aucune intervention externe en fournissant les meilleures propriétés possibles dans chaque environnement.

Algorithmes spéculatifs. En informatique distribuée, l'approche classique pour évaluer l'efficacité d'un algorithme est de considérer le cas le plus défavorable afin de fournir une borne supérieure (i.e., la performance maximale selon certaines métriques comme le temps utilisé, la mémoire utilisée, le nombre de messages envoyés, *etc.*, nécessaire pour résoudre un problème) pour n'importe quelle exécution possible dans un environnement donné. L'approche spéculative [114, 77, 9] se fonde sur l'observation que le pire environnement possible dans lequel un problème peut être résolu n'est pas toujours le plus fréquent. Lorsque l'environnement dans lequel les algorithmes sont le plus souvent exécutés ne correspond pas au cas le plus défavorable possible, l'approche classique (considérant le cas le plus défavorable) peut fournir des bornes supérieures qui sont très éloignées des bornes obtenues en "pratique". Pour pallier ce problème, l'approche spéculative consiste à optimiser les algorithmes pour l'environnement le plus fréquent dans lequel ils seront exécutés. L'analyse de l'efficacité d'un algorithme spéculatif se fait toujours par rapport au cas le plus défavorable de manière à fournir une borne supérieure pour toute exécution dans l'environnement le plus probable.

Plus précisément, les algorithmes spéculatifs résolvent le problème étudié dans tous les environnements dans lesquels ils sont susceptibles d'être exécutés, mais aussi ils sont plus efficaces lorsqu'ils sont exécutés dans les environnements les plus probables en pratique. L'optimisation fournie par un algorithme spéculatif est, selon le contexte, fonction d'une complexité temps, mémoire, en nombre de messages échangés...

Cette approche permet de contourner d'importantes bornes inférieures obtenues dans certains environnements rares puisque les algorithmes spéculatifs fournissent une solution optimisée au problème étudié lorsqu'ils sont exécutés dans des environnements dans lesquels le problème est fréquemment exécuté (i.e., la borne supérieure obtenue dans les environnements les plus fréquents est inférieure à la borne inférieure obtenue en considérant toutes les exécutions possibles).

Réseaux de robots dans des environnements dynamiques. Dans cette thèse, nous nous intéressons aux réseaux de robots [136], i.e., réseaux constitués de processus capables de bouger et dotés de capteurs pour détecter leur environnement. Nous considérons une cohorte de robots avec peu de capacités. De nombreuses applications existent pour de tels systèmes multi-robots: encerclement, patrouille, exploration de divers environnements... Les capacités de déplacement et sensorielles sont des ingrédients clés pour réaliser des tâches collaboratives dans de tels systèmes distribués. Dans cette thèse, nous considérons des systèmes distribués avec des hypothèses et des capacités de robot aussi faibles que possible et nous déterminons la faisabilité de problèmes dans des modèles donnés.

Une première approche "naturelle" lors de l'étude des réseaux de robots consiste à considérer des robots évoluant dans l'espace continu [45, 72, 139, 123]. Une seconde approche consiste à considérer des robots évoluant dans des environnements discrets modélisés par des graphes statiques où les nœuds représentent les emplacements où les robots peut être situés et les arêtes représentent la possibilité pour un robot de bouger d'un emplacement à un autre [85, 17, 64, 69].

Cependant, tous les environnements ne sont pas statiques : certains sont dynamiques et peuvent être représentés par des graphes dynamiques dans lesquels les nœuds et les arêtes peuvent apparaître et disparaître avec le temps. Les graphes dynamiques sont utiles pour représenter des environnements instables susceptibles de changer au fil du temps, comme par exemple un réseau de transport, un bâtiment dans lequel les portes sont fermées et ouvertes au cours du temps, ou des rues qui sont fermées au cours du temps en raison de travaux en cours ou d'embouteillage dans une ville.

Dans cette thèse, nous considérons des robots évoluant dans des graphes dynamiques composés d'un ensemble de nœuds statiques qui modélisent les emplacements où les robots peuvent

être situés, et d'un ensemble d'arêtes qui peuvent apparaître et disparaître avec le temps et représentent la possibilité pour un robot de se déplacer d'un emplacement à un autre à un moment donné.

Il existe plusieurs systèmes modélisables grâce aux graphes dynamiques, tous ces systèmes n'ont pas la même connectivité dans le temps (i.e., fréquences d'apparition et de disparition des arêtes). Par exemple, les réseaux de transport en commun ont une connectivité périodique induite par les mouvements des transporteurs tandis que les réseaux téléphoniques ont une connectivité incontrôlée. Par conséquent, pour représenter les différentes hypothèses de connectivité, il existe différentes classes de graphes dynamiques [40, 38].

Lorsque la vitesse et la planification des modifications topologiques varient, certains problèmes deviennent impossibles ou les bornes inférieures pour résoudre certains problèmes augmentent. Cela motive l'utilisation des approches progressivement dégradantes et spéculatives.

Problèmatiques de la thèse. Comme les approches progressivement dégradantes et spéculatives n'ont jamais été appliquées aux réseaux de robots, l'objectif poursuivi dans cette thèse est de déterminer si cela est faisable et si cela en vaut la peine.

Au premier abord, les fortes restrictions de ce modèle original ne plaident pas en faveur d'une réponse positive. En particulier, dans un système distribué traditionnel, les processus échangent des informations en envoyant des messages (qui peuvent être dupliqués pour surmonter les absences temporaires ou permanentes des arêtes sans aucun impact sur le comportement en cours des processus), alors que les robots doivent se déplacer pour communiquer (directement ou non) ensemble (sans la possibilité de se dupliquer, ce qui les oblige à gérer leurs mouvements en fonction de la dynamique de l'environnement). Par conséquent, concevoir un algorithme pour un réseau de robots capables de s'adapter à plusieurs environnements semble être une tâche ardue.

La contribution de cette thèse est de relever ce défi et de fournir une réponse positive à cette question en mettant l'accent sur deux problèmes classiques de la littérature: le rassemblement (où tous les robots doivent être situés, en temps fini et borné, sur le même noeud du graphe) et l'exploration perpétuelle (où tous les nœuds du graphe doivent être visités infiniment souvent par au moins un robot).

Contributions de la thèse. Nous montrons tout d'abord Section B.2 qu'il est possible d'appliquer une approche progressivement dégradante aux réseaux de robots. Pour cela nous étudions le problème du rassemblement dans des anneaux dynamiques. Ce problème est impossible à résoudre dans des anneaux très dynamiques (anneaux dynamiques dans lesquels au plus une arête peut être manquante pour toujours à partir d'un certain temps). Ainsi ce problème est un bon cas d'étude pour une approche progressivement dégradante. Les résultats présentés dans cette section ont fait l'objet d'une publication dans *International Symposium on Stabilization, Safety, and Security of Distributed Systems* 2018 [31].

Nous montrons ensuite, Section B.3, qu'il est possible d'appliquer une approche spéculative aux réseaux de robots. Pour cela nous étudions l'exploration perpétuelle dans des anneaux dynamiques. Nous considérons ce problème en utilisant des robots non sujets aux fautes ainsi qu'en présence de fautes transitoires (fautes arbitraires qui ne sont plus présentes à partir d'un certain temps). Les résultats obtenus en considérant des robots sans faute ont été publiés dans *IEEE International Conference on Distributed Computing Systems* 2017 [29], et dans *rencontres francophones sur les Aspects Algorithmiques des Telecommunications* 2017 [30]. Ceux obtenus en considérant des robots sujets aux fautes transitoires ont été publiés dans *International Symposium on Stabilization, Safety, and Security of Distributed Systems* 2016 [27], et dans *Theoretical Computer Science* [28].

Nous nous sommes également intéressés au problème de l'approche dans le plan (qui consiste pour deux robots, dotés d'une vision limitée, à être localisés, en temps fini, dans le champ de vision l'un de l'autre). Nous avons fourni un algoirthme en temps polynomial (en termes de

mouvements) qui résout ce problème en utilisant des robots asynchrones dotés de faibles capacités pour s'orienter dans le plan. Ce travail a été publié dans *International Symposium on Distributed Computing* 2017 [22], dans *Distributed Computing* [24], et dans *rencontres francophones sur les Aspects Algorithmiques des Telecommunications* 2018 [23]. L'algorithme présenté dans ces articles n'étant ni progressivement dégradant ni spéculatif, nous ne le détaillerons pas ici.

B.1 Contexte

Nous considérons un ensemble de robots autonomes se déplaçant dans un environnement discret, anonyme, bidirectionnel et dynamique. Par environnement discret, anonyme et bidirectionnel, nous désignons un espace partitionné en un nombre fini d'emplacements (discret) représenté par un graphe où chaque nœud est indistinguable des autres nœuds (anonyme) et représente un emplacement susceptible d'être occupé par un ou plusieurs robots et où une arête représente la possibilité pour un robot de se déplacer (dans les deux directions) d'un emplacement à un autre (bidirectionnel). De plus, ce graphe est dynamique dans le sens où ses arêtes peuvent apparaître et disparaître au cours du temps. Pour modéliser cela, nous considérons le temps comme discret et définissons un graphe évolutif (selon [80]) comme une suite de graphes statiques construits sur un ensemble constant de nœuds. Chaque graphe de cette suite contient les arêtes présentes dans l'environnement des robots à l'instant associé. Dans un graphe évolutif, les arêtes peuvent se repartir en deux catégories : les arêtes manquantes à terme et les arêtes récurrentes. Les premières sont des arêtes qui disparaissent définitivement après un instant donné; les secondes sont infiniment souvent présentes dans la suite de graphes.

Il existe plusieurs types de graphes dynamiques. Nous présentons ici ceux que nous considérons dans la thèse. Par abus de langage, nous dirons qu'un graphe évolutif est "X" si ce graphe appartient au type X de graphes dynamiques. Un graphe évolutif est dit statique (noté $\mathcal{ST}$) si toutes ses arêtes sont présentes à chaque instant. Un graphe évolutif est récurrent borné (noté $\mathcal{BRE}$) si chacune de ses arêtes est présente au moins une fois toutes les δ unités de temps. Un graphe évolutif est récurrent (noté $\mathcal{RE}$) si chacune de ses arêtes est présente infiniment souvent. Un graphe évolutif est dit toujours connecté (noté $\mathcal{AC}$) si à chaque instant, les arêtes présentes forment un graphe connexe. Enfin un graphe évolutif est connecté dans le temps (noté $\mathcal{COT}$) si ses arêtes peuvent apparaître et disparaître de manière imprévisible sans aucune hypothèse de récurrence, de stabilité ou de périodicité à travers le temps, à la condition que les arêtes récurrentes induisent un graphe couvrant connexe. Cette condition permet à un robot d'atteindre tout nœud du graphe quel que soit son nœud et sa date de départ. Notons qu'il existe des relations d'inclusion entre ces différents types de graphes dynamiques : $\mathcal{ST} \subset \mathcal{BRE} \subset \mathcal{RE} \subset \mathcal{COT}$ et $\mathcal{ST} \subset \mathcal{AC} \subset \mathcal{COT}$.

Chaque robot est doté d'une unité de calcul, d'un mécanisme lui permettant de se déplacer d'un nœud à un autre et de capteurs capables de déterminer la présence des arêtes adjacentes au nœud sur lequel le robot se trouve. Les capteurs donnent des informations aux robots, comme par exemple, la multiplicité faible (c'est-à-dire de savoir si le robot est seul à occuper le nœud ou non) ou forte (c'est-à-dire de connaître le nombre exact de robots se trouvant sur le nœud du robot courant) du nœud. Si plusieurs robots occupent un même nœud au même instant, nous dirons qu'ils forment une tour. Les robots sont dotés d'une mémoire persistante et évoluent dans un graphe dynamique de manière synchrone en effectuant chacun un cycle atomique à chaque instant. Ce cycle comporte trois phases. La première est la phase d'observation. Elle consiste à capter et transmettre ces informations (présence des arêtes adjacentes, la multiplicité du nœud, *etc.*) à l'unité de calcul du robot. Elle est suivie d'une phase de calcul au cours de laquelle, sur la base des observations recueillies, l'unité de calcul exécute un algorithme qui établit la direction (un numéro de port d'une arête) à prendre lors de la phase de déplacement, c'est-à-dire l'arête adjacente que doit essayer de traverser (de manière atomique) le robot. Si l'arête n'est pas présente à cet instant, le robot reste sur son nœud.

B.2 Approche progressivement dégradante

Il n'existe pas d'algorithme progressivement dégradant utilisant des robots. Le problème du rassemblement étant impossible à résoudre dans certains types de graphes dynamiques, nous avons choisi de nous intéresser à ce problème dans les graphes dynamiques et de lui trouver une solution progressivement dégradante lorsque la dynamicité du graphe augmente.

État de l'art. Il existe dans la littérature des algorithmes résolvant le rassemblement dans les graphes dynamiques (mais comme indiqué, ils ne sont pas progressivement dégradants). Notamment, ces algorithmes ne considèrent qu'un seul environnement modélisé par un seul type de graphes dynamiques. Parmi ces algorithmes, celui de Yamauchi et al. [142] résout le problème du rendez-vous (rassemblement de deux robots) dans des anneaux dynamiques où à chaque instant de temps la présence de chaque arête est décidé avec une certaine probabilité. Izumi et al. [110] eux s'intéressent au problème du rassemblement dans des anneaux $\mathcal{RE}$. Di Luna et al. [122] se sont intéressés au problème du rassemblement dans des anneaux $\mathcal{AC}$. Ils ont prouvé que ce problème est impossible à résoudre dans ce type d'anneaux dynamiques et ont alors donné un algorithme résolvant le rassemblement approché avec terminaison (où tous les robots doivent, en temps fini et borné, terminer leur exécution sur deux nœuds adjacents de l'anneau dyamique) dans des anneaux $\mathcal{AC}$. Ooshita et Datta [128] se sont intéressés aux anneaux $\mathcal{COT}$. Le résultat d'impossibilité de Di Luna et al. [122] est également valide pour les anneaux $\mathcal{COT}$. Ooshita et Datta [128] ont fourni un algorithme résolvant le rassemblement approché sans terminaison (où tous les robots doivent, en temps fini, se retrouver sur deux nœuds adjacents de l'anneau dynamique sans nécessairement terminer leur exécution) dans des anneaux $\mathcal{COT}$.

Nos résultats. L'impossibilité énoncée par Di Luna et al. [122] a motivé la conception de notre algorithme progressivement dégradant résolvant le rassemblement dans des anneaux dynamiques. C'est le premier algorithme progressivement dégradant appliqué aux réseaux de robots. L'algorithme que nous avons conçu résout le rassemblement dans les graphes dynamiques où ce problème est résoluble, et le dégrade (résout des versions plus faibles) lorsqu'il est exécuté dans des graphes dynamiques où ce problème est impossible. Nous avons un seul algorithme qui s'exécute et qui s'adapte à la dynamicité courante sans pour autant avoir un mécanisme de détection de la dynamicité.

La sûreté du rassemblement impose que tous les robots finissent leur exécution sur le même nœud, et la vivacité de ce problème impose que tous les robots finissent en temps fini et borné. Les problèmes plus faibles que nous proposons dégradent la vivacité du rassemblement et conservent sa sûreté. Plus précisément, nous affaiblissons le problème du rassemblement tel que au plus un robot peut ne pas terminer son exécution ou (non exclusif) tous les robots qui terminent le font en temps fini (mais non borné).

Ainsi, nous proposons 3 variantes du rassemblement: le rassemblement ultime (où tous les robots terminent leur exécution sur le même nœud du graphe en temps fini), le rassemblement faible (au moins tous les robots sauf un terminent leur exécution sur le même nœud du graphe en temps fini et borné), et le rassemblement faible ultime (au moins tous les robots sauf un terminent leur exécution sur le même nœud du graphe en temps fini). Chacune des ces variantes préserve la sûreté du rassemblement et garantit qu'au moins tous sauf un robots terminent (ultimement ou en temps borné, en fonction de la variante du rassemblement considérée) leur exécution sur le même nœud de l'anneau.

Notre algorithme résout le rassemblement dans les anneaux $\mathcal{BRE}$ et $\mathcal{ST}$, le rassemblement ultime dans les anneaux $\mathcal{RE}$, le rassemblement faible dans les anneaux $\mathcal{AC}$ et le rassemblement faible ultime dans les anneaux $\mathcal{COT}$. Les propriétés définissant le type d'un graphe dynamique ne peuvent être vérifiées que de manière ultime, ainsi une des difficultés d'un tel algorithme est de réussir à résoudre la bonne version du rassemblement sans être capable de détecter le type

	rassemblement	rassemblement ultime	rassemblement faible	rassemblement faible ultime
$\mathcal{COT}$	Impossible	Impossible	Impossible	Possible
$\mathcal{AC}$	Impossible	Impossible	Possible	—
$\mathcal{RE}$	Impossible	Possible	Impossible	—
$\mathcal{BRE}$	Possible	—	—	—
$\mathcal{ST}$	Possible	—	—	—

Table B.1: Résumé de nos résultats. Le symbole — signifie qu'une variante plus forte du problème est déjà résolue sous les hypothèses de dynamicité. Notre algorithme est progressivement dégradant vu qu'il résout chaque variante du rassemblement dés que les hypothèses de dynamicité le permettent.

de graphes dynamiques dans lequel il est exécuté. Parmi les types d'anneaux dynamiques que nous étudions, certains peuvent avoir des arêtes manquantes à terme, ce qui ajoute des difficultés dans la manière d'appréhender le problème du rassemblement en fonction du type de graphes dynamiques considéré.

Pour fonctionner notre algorithme a besoin d'au moins 4 robots. Ces robots sont synchrones, connaissent la taille de l'anneau, et le nombre total de robots dans le système. Chaque robot possède un identifiant (positif) distinct. Initiallement, un robot ne connaît que la valeur de son identifiant. Les robots sont capables de communiquer en lisant les valeurs des variables des robots qui sont situés sur le même nœud qu'eux. Ils possèdent la détection de la multiplicité forte (ils sont capables de connaître le nombre exact de robots situés sur le même nœud qu'eux). Ils ont la même chiralité (ils ont accès à la même numérotation des ports sur les nœuds, leur permettant alors d'avoir la même orientation dans l'anneau). Notre algorithme a besoin de tels robots pour fonctionner correctement, cependant la nécessité de ces hypothèses reste une question ouverte.

Nous avons également prouvé que notre algorithme résout la meilleure version du rassemblement (parmi celles que nous considérons) dans les graphes dynamiques que nous étudions, notamment en prouvant, sous les hypothèses présentées ci-dessus, une série de résultats d'impossibilité que nous résumons dans le Tableau B.1. Ainsi en plus d'être progressivement dégradant, nous pouvons qualifier notre algorithme de progressivement dégradant de manière optimale.

Cependant, cela ne signifie pas qu'il n'existe pas un algorithme satisfaisant des variantes plus fortes du rassemblement parmi l'infinité de versions de ce problème qu'il est possible de proposer. Nous pouvons qualifier d'optimum un algorithme progressivement dégradant résolvant le rassemblement s'il résout les meilleures versions du rassemblement dans les types de graphes dynamiques analysés. Une question ouverte intéressante à creuser est de savoir si un tel algorithme existe.

De même, sans considérer le problème du rassemblement, étudier des algorithmes progressivement dégradants optimums semble être une tâche ardue.

B.3 Approche spéculative

Il n'existe pas d'algorithme spéculatif utilisant des robots. Nous avons choisi de nous intéresser au problème de l'exploration afin de proposer une approche spéculative à ce problème dans les graphes dynamiques.

État de l'art. Il existe dans la littérature des algorithmes résolvant l'exploration dans des graphes dynamiques (mais comme indiqué, ils ne sont pas spéculatifs vu qu'ils étudient le problème dans un type de graphes dynamiques précis). Parmi ces algorithmes, certains considèrent que les robots connaissent la dynamique du graphe : ils savent quelles arêtes sont présentes et à quels instants. C'est notamment le cas de Ilcinkas et Wade [106] qui se sont intéressés à l'exploration dans des anneaux T-intervalle connectés (qui sont des graphes dynamiques tels que

à chaque instant de temps les arêtes présentes forment un graphe connexe et pour tout intervalle de temps de taille T, un sous-ensemble des arêtes présentes forment le même arbre couvrant), et de Ilcinkas et al. [104] qui ont étudié l'exploration dans des cactus (arbres de cycles) T-intervalle connectés. D'autres articles par contre considèrent que les robots n'ont pas de connaissance quant à la dynamique du graphe. Parmi ces algorithmes, ceux de Flocchini et al. [89, 88, 87], et de Wade et Ilcinkas [105] résolvent le problème de l'exploration avec arrêt (où les robots doivent explorer le graphe en temps fini et doivent arrêter leur exécution également en temps fini une fois qu'ils ont détecté qu'ils avaient exploré le graphe) dans des graphes périodiques (graphes dynamiques dans lesquels les arêtes sont présentes de manière périodique). Ilcinkas et Wade [106] ont étudié l'exploration dans des anneaux $\mathcal{BRE}$ et T-intervalle connectés, i.e., même si au plus une arête de l'anneau peut être manquante à chaque instant de temps, chaque arête est au moins présente une fois toutes les δ unités de temps. Flocchini et al. [86] ont considéré le problème de l'exploration avec arrêt dans des graphes $\mathcal{AC}$ particuliers où lorsqu'une arête disparaît elle ne réapparaît jamais. Di Luna et al. [125] ont eux étudié l'exploration avec arrêt des anneaux $\mathcal{AC}$. Finalement, Gotoh et al. [99] ont analysé l'exploration avec arrêt de tori (grille où chaque nœud extrêmité est adjacent au nœud extrêmité opposé de la même ligne ou colonne de la grille, si il existe) où chaque anneau est un graphe $\mathcal{AC}$.

Nos résultats. Nous nous sommes intéressés au problème de l'exploration perpétuelle, et nous avons fourni des algorithmes spéculatifs à ce problème en considérant des robots non sujets aux fautes, ainsi que des robots sujets aux fautes transitoires.

Sans faute. Bien que le problème de l'exploration ait été étudié massivement dans les graphes statiques, et aussi dans les graphes dynamiques, il n'a jamais été étudié dans les graphes dynamiques $\mathcal{COT}$. Dans les anneaux $\mathcal{COT}$, le temps de couverture (pire temps du temps minimal pris par les robots pour explorer au moins une fois chaque nœud de l'anneau depuis n'importe quel temps de toutes les exécutions possibles de leur algorithme) est non borné vu qu'il est possible dans ce type d'anneau d'avoir toutes les arêtes manquantes et cela pendant un temps arbitrairement long. Cela motive la conception d'algorithmes spéculatifs résolvant l'exploration perpétuelle dans les graphes dynamiques.

Dans un premier temps, nous nous sommes intéressés au problème de l'exploration perpétuelle dans les anneaux $\mathcal{COT}$. Nous avons considéré des robots avec peu de capacités évoluant dans ce type de graphes dynamiques : ils sont anonymes (n'ont pas d'identifiant), synchrones, n'ont pas la même chiralité (les robots n'ont pas nécessairement accès à une numérotation des ports commune, ils ne peuvent donc pas s'orienter dans l'anneau grâce à une telle numérotation), ils n'ont aucune connaissance sur le graphe (ni la taille, ni la dynamicité, *etc.*), ils ne peuvent pas communiquer de manière directe, ils ont la détection de la multiplicité faible (un robot est capable de savoir s'il est seul ou non sur son nœud, mais ne peut pas connaître le nombre exact de robot situé sur le même nœud que lui).

Sous ces hypothèses, nous avons caractérisé le nombre nécessaire et suffisant de robots permettant de résoudre le problème de l'exploration perpétuelle. Nous avons montré que 2 robots ne peuvent pas résoudre l'exploration perpétuelle dans les anneaux $\mathcal{COT}$ de tailles 4 ou plus, et que 1 robot ne peut pas résoudre l'exploration perpétuelle dans les anneaux $\mathcal{COT}$ de tailles 3 ou plus. Les résultats obtenus sont résumés dans le Tableau B.2. En fonction de ces résultats, nous avons fourni des algorithmes pour montrer que le nombre nécessaire de robots est également suffisant pour explorer de manière perpétuelle les anneaux $\mathcal{COT}$. Notre algorithme utilisant un nombre de robots supérieur ou égal à 3 est spéculatif. En effet, il résout l'exploration perpétuelle dans les anneaux $\mathcal{COT}$, et est optimisé pour les anneaux $\mathcal{ST}$: son temps de couverture dans les anneaux $\mathcal{COT}$ n'est pas borné, mais il est borné et même asymptotiquement optimal dans les anneaux $\mathcal{ST}$ (son temps de couverture pour les anneaux $\mathcal{ST}$ est en $\Theta(n - R)$ unités de temps, où n est la taille de l'anneau et $\mathcal{R}$ est le nombre de robots dans le système).

Nombre de robots	Taille des anneaux	Résultats
3 et plus	≥ 4	Possible
2	> 3	Impossible
	$= 3$	Possible
1	> 2	Impossible

Table B.2: Résumé des résultats en considérant des robots non sujets aux fautes.

Avec faute. Dans le paragraphe précédent, nous avons présenté des résultats avec des robots possédant de faibles capacités. Moins les robots ont de capacités moins ils sont sujets aux fautes (vu que les capacités des robots sont induites par des mécanismes qui peuvent tomber en panne). Cependant, bien que les robots aient peu de capacités, ils peuvent tout de même être sujets à des fautes. Dans ce paragraphe, nous étudions en particulier des robots pouvant être sujets à des fautes transitoires qui sont des fautes arbitraires telles qu'il existe un temps à partir duquel elles ne sont plus présentes. Les algorithmes qui tolèrent les fautes transitoires sont dits auto-stabilisants : un algorithme auto-stabilisant est défini par Dijkstra [73] comme étant un algorithme qui retrouve un comportement correct en temps fini quelque soit la configuration initiale du système (qui capture les effets des fautes transitoires dans le système). Quand un algorithme auto-stabilisant utilise des robots, ces robots sont également dits auto-stabilisants. Les algorithmes auto-stabilisants sont des algorithmes puissants puisqu'ils résolvent le problème étudié même après que de gros dommages aient eu lieu dans le système. Il n'y a aucun algorithme auto-stabilisant qui résolve le problème de l'exploration ni dans les graphes statiques ni dans les graphes dynamiques.

Comme indiqué, un algorithme auto-stabilisant doit résoudre le problème étudié quelque soit la configuration initiale du système. Ainsi, pour être auto-stabilisant un algorithme d'exploration perpétuelle doit tolérer n'importe quelle position initiale des robots. En particulier, il doit tolérer que tous les robots commencent leur exécution sur le même nœud. Si plusieurs robots non identifiés commencent leur exécution sur le même nœud, par déterminisme, ils vont se comporter comme un seul et même robot. Or, comme indiqué dans le paragraphe précédent, il n'est pas possible de résoudre le problème de l'exploration perpétuelle dans les anneaux $\mathcal{COT}$ avec un seul robot. Ainsi, pour résoudre le problème de l'exploration perpétuelle de manière auto-stabilisante les robots doivent être dotés d'identifiants distincts.

Nous nous sommes intéressés au problème de l'exploration perpétuelle utilisant des robots auto-stabilisants. Nous avons tout d'abord considéré ce problème pour des robots évoluant dans les anneaux $\mathcal{COT}$. Les robots auto-stabilisants que nous utilisons ont les capacités suivantes : ils sont synchrones, ils n'ont pas la même chiralité, ils ont des identifiants distincts mais ne connaissent que leur propre identifiant, ils n'ont pas de connaissance du graphe dynamique, ils ne peuvent pas communiquer directement, ils sont dotés de la détection de la multiplicité forte (ils sont capables de connaître le nombre exact de robots qui se situent sur le même nœud qu'eux). Vu que se sont des robots auto-stabilisants, leur mémoire persistante est divisée en deux parties : une corruptible contenant les variables et une incorruptible contenant les constantes (telles que leur identifiant) et leur code.

De la même manière que précédemment, sous les hypothèses citées ci-dessus, nous avons caractérisé le nombre nécessaire et suffisant de robots auto-stabilisants permettant de résoudre le problème de l'exploration perpétuelle. Nous avons montré que 2 robots ne peuvent pas résoudre l'exploration perpétuelle dans les anneaux de tailles 4 ou plus, et que 1 robot ne peut pas résoudre l'exploration perpétuelle dans les anneaux de tailles 3 ou plus. Les résultats de nécessité montrés ici ne sont pas impliqués par ceux considérant des robots sans faute qui eux n'ont pas d'identifiants. Les résultats obtenus sont résumés dans le Tableau B.3. En fonction de ces résultats, nous avons fourni des algorithmes pour montrer que le nombre nécessaire de

Nombre de robots	Taille des anneaux	Résultats
3	≥ 4	Possible
2	> 3	Impossible
	$= 3$	Possible
1	> 2	Impossible

Table B.3: Résumé des résultats en considérant des robots sujets aux fautes transitoires.

robots auto-stabilisants est également suffisant pour explorer de manière perpétuelle les anneaux $\mathcal{COT}$. Notre algorithme utilisant un nombre de robots égal à 3 est spéculatif. En effet, il résout l'exploration perpétuelle dans les anneaux $\mathcal{COT}$, et est optimisé pour les anneaux $\mathcal{ST}$: son temps de couverture dans les anneaux $\mathcal{COT}$ n'est pas borné, mais il est borné et même asymptotiquement optimal dans les anneaux $\mathcal{ST}$ (son temps de couverture pour les anneaux $\mathcal{ST}$ est en $\Theta(n)$ unités de temps, où n est la taille de l'anneau).

Nous pouvons noter que nos deux algorithmes spéculatifs (celui utilisant des robots sans faute, comme celui utilisant des robots sujets aux fautes transitoires) sont spéculatifs de manière optimale : ils fournissent le meilleur temps de couverture parmi ceux proposés (non borné et borné) dans les anneaux $\mathcal{ST}$ et $\mathcal{COT}$.

Cependant, cette définition d'algorithmes spéculatifs de manière optimale n'exclut pas la possibilité qu'un algorithme soit spéculatif optimal sans pour autant résoudre le problème étudié avec les meilleures performances possibles. Nous pouvons alors qualifier un algorithme spéculatif d'optimum s'il résout le problème étudié avec les meilleures performances possibles dans chacun des types de graphes dynamiques considérés. Analyser de tels algorithmes semble être une future tâche attrayante.

B.4 Conclusion

Dans cette thèse nous avons appliqué les approches progressivement dégradante et spéculative aux réseaux de robots évoluant dans des graphes dynamiques. Pour cela, nous avons étudié le rassemblement de manière progressivement dégradante et l'exploration perpétuelle de manière spéculative. De multiple perspectives peuvent être envisagées, notamment il serait intéressant de trouver de nouvelles mesures de complexités (en temps et en terme de mouvements) pouvant s'appliquer aux réseaux de robots évoluant dans des graphes dynamiques pour améliorer la conception d'algorithmes spéculatifs utilisant de tels robots. Aussi, une perspective qui pourrait être envisagée serait de combiner les approches progressivement dégradante et spéculative. En effet, il serait intéressant d'étudier l'aspect spéculatif d'un algorithme progressivement dégradant : pour chaque ensemble de types de graphes dynamiques dans lesquels un algorithme progressivement dégradant résout un certain problème, l'optimiser pour le type de graphes dynamiques où il sera le plus souvent exécuté.

Bibliography

[1] E. Aaron, D. Krizanc, and E. Meyerson. DMVP: foremost waypoint coverage of time-varying graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 29–41, 2014.

[2] S. Abshoff, A. Cord-Landwehr, M. Fischer, D. Jung, and F. Meyer auf der Heide. Gathering a closed chain of robots on a grid. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 689–699, 2016.

[3] A. Agarwalla, J. Augustine, W. K. Moses Jr., S. Madhav K., and A. Krishna Sridhar. Deterministic dispersion of mobile robots in dynamic rings. In *International Conference on Distributed Computing and Networking (ICDCN)*, pages 19:1–19:4, 2018.

[4] C. Agathangelou, C. Georgiou, and M. Mavronicolas. A distributed algorithm for gathering many fat mobile robots in the plane. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 250–259, 2013.

[5] D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers. Generating fast indulgent algorithms. *Theory of Computing Systems (Theor. Comput. Sci.)*, 51(4):404–424, 2012.

[6] A. Aljohani and G. Sharma. Complete visibility for mobile robots with lights tolerating faults. *International Journal of Networking and Computing (IJNC)*, 8(1):32–52, 2018.

[7] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation (IEEE Trans. Robotics and Automation)*, 15(5):818–828, 1999.

[8] H. Attiya and J. Welch. *Fundamentals, Simulations and Advances Topics (2nd edition).* John Wiley Interscience, 2004.

[9] P.-L. Aublin, R. Guerraoui, N. Knezevic, V. Quéma, and M. Vukolic. The next 700 BFT protocols. *ACM Transactions on Computer Systems (ACM Trans. Comput. Syst.)*, 32(4):12:1–12:45, 2015.

[10] J. Augustine and W. K. Moses Jr. Dispersion of mobile robots: A study of memory-time trade-offs. In *International Conference on Distributed Computing and Networking (ICDCN)*, pages 1:1–1:10, 2018.

[11] C. Avin, M. Koucký, and Z. Lotker. Cover time and mixing time of random walks on dynamic graphs. *Random Structures and Algorithms (Random Struct. Algorithms)*, 52(4):576–596, 2018.

[12] R. Baldoni, F. Bonnet, A. Milani, and M. Raynal. On the solvability of anonymous partial grids exploration by mobile robots. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 428–445. Springer, 2008.

[13] E. Bampas, J. Czyzowicz, L. Gasieniec, David Ilcinkas, and Arnaud Labourel. Almost optimal asynchronous rendezvous in infinite multidimensional grids. In *International Symposium on Distributed Computing (DISC)*, pages 297–311, 2010.

[14] L. Barrière, P. Flocchini, E. Mesa Barrameda, and N. Santoro. Uniform scattering of autonomous mobile robots in a grid. *International Journal of Foundations of Computer Science (Int. J. Found. Comput. Sci.)*, 22(3):679–697, 2011.

[15] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Rendezvous and election of mobile agents: Impact of sense of direction. *Theory of Computing Systems (Theory Comput. Syst.)*, 40(2):143–162, 2007.

[16] S. Beamer, A. Buluç, K. Asanovic, and D. A. Patterson. Distributed memory breadth-first search revisited: Enabling bottom-up search. In *IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum*, pages 1618–1627, 2013.

[17] M. A. Bender and D. K. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. In *Symposium on Foundations of Computer Science (FOCS)*, pages 75–85, 1994.

[18] K. A. Berman. Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks*, 28(3):125–134, 1996.

[19] S. Bhagat, S. Gan Chaudhuri, and K. Mukhopadhyaya. Fault-tolerant gathering of asynchronous oblivious mobile robots under one-axis agreement. *Journal of Discrete Algorithms (J. Discrete Algorithms)*, 36:50–62, 2016.

[20] M. Biely, P. Robinson, U. Schmid, M. Schwarz, and K. Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *Theoretical Computer Science (Theor. Comput. Sci.)*, 726:41–77, 2018.

[21] L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil. Exclusive perpetual ring exploration without chirality. In *International Symposium on Distributed Computing (DISC)*, pages 312–327, 2010.

[22] S. Bouchard, M. Bournat, Y. Dieudonné, S. Dubois, and F. Petit. Asynchronous approach in the plane: A deterministic polynomial algorithm. In *International Symposium on Distributed Computing (DISC)*, pages 8:1–8:16, 2017.

[23] S. Bouchard, M. Bournat, Y. Dieudonné, S. Dubois, and F. Petit. Approche asynchrone dans le plan : un algorithme déterministe polynomial. In *rencontres francophones sur les Aspects Algorithmiques des Telecommunications (ALGOTEL)*, 2018.

[24] S. Bouchard, M. Bournat, Y. Dieudonné, S. Dubois, and F. Petit. Asynchronous approach in the plane: A deterministic polynomial algorithm. *Distributed Computing*, to appear.

[25] S. Bouchard, Y. Dieudonné, and B. Ducourthial. Byzantine gathering in networks. *Distributed Computing*, 29(6):435–457, 2016.

[26] S. Bouchard, Y. Dieudonné, and A. Lamani. Byzantine gathering in polynomial time. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 147:1–147:15, 2018.

[27] M. Bournat, A. K. Datta, and S. Dubois. Self-stabilizing robots in highly dynamic environments. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 54–69, 2016.

[28] M. Bournat, A. K. Datta, and S. Dubois. Self-stabilizing robots in highly dynamic environments. *Theoretical Computer Science (Theor. Comput. Sci.)*, to appear.

[29] M. Bournat, S. Dubois, and F. Petit. Computability of perpetual exploration in highly dynamic rings. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 794–804, 2017.

[30] M. Bournat, S. Dubois, and F. Petit. Quel est le nombre optimal de robots pour explorer un anneau hautement dynamique ? In *rencontres francophones sur les Aspects Algorithmiques des Telecommunications (ALGOTEL)*, 2017.

[31] M. Bournat, S. Dubois, and F. Petit. Gracefully degrading gathering in dynamic rings. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, page to appear, 2018.

[32] Z. Bouzid and A. Lamani. Robot networks with homonyms: The case of patterns formation. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 92–107, 2011.

[33] Z. Bouzid, M. Gradinariu Potop-Butucaru, and S. Tixeuil. Byzantine convergence in robot networks: The price of asynchrony. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 54–70, 2009.

[34] N. Braud-Santoni, S. Dubois, M.-H. Kaaouachi, and F. Petit. The next 700 impossibility results in time-varying graphs. *International Journal of Networking and Computing (IJNC)*, 6(1):27–41, 2016.

[35] D. Canepa, X. Défago, T. Izumi, and M. Potop-Butucaru. Flocking with oblivious robots. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 94–108, 2016.

[36] D. Canepa and M. Gradinariu Potop-Butucaru. Stabilizing flocking via leader election in robot networks. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 52–66, 2007.

[37] R. Carli and F. Bullo. Quantized coordination algorithms for rendezvous and deployment. *SIAM Journal on Control and Optimization (SIAM J. Control and Optimization)*, 48(3):1251–1274, 2009.

[38] A. Casteigts. Finding structure in dynamic networks. *CoRR*, abs/1807.07801, 2018.

[39] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro. Shortest, fastest, and foremost broadcast in dynamic networks. *International Journal of Foundations of Computer Science (Int. J. Found. Comput. Sci.)*, 26(4):499–522, 2015.

[40] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)*, 27(5):387–408, 2012.

[41] J. Chalopin, Y. Dieudonné, A. Labourel, and A. Pelc. Rendezvous in networks in spite of delay faults. *Distributed Computing*, 29(3):187–205, 2016.

[42] J. Chalopin, P. Flocchini, B. Mans, and N. Santoro. Network exploration by silent and oblivious robots. In *Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 208–219, 2010.

[43] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving the robots gathering problem. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1181–1196, 2003.

[44] J. Clément, X. Défago, M. Gradinariu Potop-Butucaru, T. Izumi, and S. Messika. The cost of probabilistic agreement in oblivious robot networks. *Information Processing Letters (Inf. Process. Lett.)*, 110(11):431–438, 2010.

[45] R. Cohen and D. Peleg. Local spreading algorithms for autonomous robot systems. *Theoretical Computer Science (Theor. Comput. Sci.)*, 399(1-2):71–82, 2008.

[46] A. Collins, J. Czyzowicz, L. Gasieniec, A. Kosowski, and R. A. Martin. Synchronous rendezvous for location-aware agents. In *International Symposium on Distributed Computing (DISC)*, pages 447–459, 2011.

[47] A. Collins, J. Czyzowicz, L. Gasieniec, and A. Labourel. Tell me where I am so I can meet you sooner. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 502–514, 2010.

[48] A. Cord-Landwehr, B. Degener, M. Fischer, M. Hüllmann, B. Kempkes, A. Klaas, P. Kling, S. Kurras, M. Märtens, F. Meyer auf der Heide, C. Raupach, K. Swierkot, D. Warner, C. Weddemann, and D. Wonisch. Collisionless gathering of robots with an extent. In *Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 178–189, 2011.

[49] A. Cord-Landwehr, M. Fischer, D. Jung, and F. Meyer auf der Heide. Asymptotically optimal gathering on a grid. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 301–312, 2016.

[50] J. Czyzowicz, L. Gasieniec, and A. Pelc. Gathering few fat mobile robots in the plane. *Theoretical Computer Science (Theor. Comput. Sci.)*, 410(6-7):481–499, 2009.

[51] J. Czyzowicz, A. Pelc, and A. Labourel. How to meet asynchronously (almost) everywhere. *ACM Transactions on Algorithms (ACM Trans. Algorithms)*, 8(4):37, 2012.

[52] S. Das, P. Flocchini, G. Prencipe, N. Santoro, and M. Yamashita. Autonomous mobile robots with lights. *Theor. Comput. Sci.*, 609:171–184, 2016.

[53] S. Das, R. Focardi, F. L. Luccio, E. Markou, D. Moro, and M. Squarcina. Gathering of robots in a ring with mobile faults. In *Italian Conference on Theoretical Computer Science (ICTCS)*, pages 122–135, 2016.

[54] S. Das, F. L. Luccio, and E. Markou. Mobile agents rendezvous in spite of a malicious agent. In *International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*, pages 211–224, 2015.

[55] A. K. Datta, A. Lamani, L. Larmore, and F. Petit. Ring exploration by oblivious agents with local vision. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 347–356, 2013.

[56] S. Datta, A. Dutta, S. Gan Chaudhuri, and K. Mukhopadhyaya. Circle formation by asynchronous transparent fat robots. In *International Conference on Distributed Computing and Internet Technology (ICDCIT)*, pages 195–207, 2013.

[57] X. Défago, M. Gradinariu, S. Messika, and P. Raipin Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering. In *International Symposium on Distributed Computing (DISC)*, pages 46–60, 2006.

[58] X. Défago and A. Konagaya. Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In *Workshop on Principles of Mobile Computing (POMC)*, pages 97–104, 2002.

[59] X. Défago and S. Souissi. Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity. *Theoretical Computer Science (Theor. Comput. Sci.)*, 396(1-3):97–112, 2008.

[60] C. Delporte-Gallet, H. Fauconnier, and H. Tran-The. Uniform consensus with homonyms and omission failures. In *International Conference on Distributed Computing and Networking (ICDCN)*, pages 161–175, 2013.

[61] D. Dereniowski and A. Pelc. Leader election for anonymous asynchronous agents in arbitrary networks. *Distributed Computing*, 27(1):21–38, 2014.

[62] A. Dessmark, P. Fraigniaud, D. R. Kowalski, and A. Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006.

[63] S. Devismes, A. Lamani, F. Petit, P. Raymond, and S. Tixeuil. Optimal grid exploration by asynchronous oblivious robots. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 64–76, 2012.

[64] S. Devismes, A. Lamani, F. Petit, and S. Tixeuil. Optimal torus exploration by oblivious robots. In *Third International Conference on Networked Systems (NETYS)*, volume 9466 of *Lecture Notes in Computer Science (LNCS)*, pages 183–199. Springer Berlin / Heidelberg, 2015.

[65] S. Devismes, F. Petit, and S. Tixeuil. Optimal probabilistic ring exploration by semi-synchronous oblivious robots. *Theoretical Computer Science (Theor. Comput. Sci.)*, 498:10–27, 2013.

[66] Y. Dieudonné, S. Dolev, F. Petit, and M. Segal. Deaf, dumb, and chatting robots, enabling distributed computation and fault-tolerance among stigmergic robot. *CoRR*, abs/0902.3549, 2009.

[67] Y. Dieudonné and A. Pelc. Deterministic polynomial approach in the plane. *Distributed Computing*, 28(2):111–129, 2015.

[68] Y. Dieudonné, A. Pelc, and D. Peleg. Gathering despite mischief. *ACM Trans. Algorithms*, 11(1):1:1–1:28, 2014.

[69] Y. Dieudonné, A. Pelc, and V. Villain. How to meet asynchronously at polynomial cost. *SIAM Journal on Computing (SIAM J. Comput.)*, 44(3):844–867, 2015.

[70] Y. Dieudonné and F. Petit. Scatter of robots. *Parallel Processing Letters*, 19(1):175–184, 2009.

[71] Y. Dieudonné and F. Petit. Self-stabilizing gathering with strong multiplicity detection. *Theoretical Computer Science (Theor. Comput. Sci.)*, 428:47–57, 2012.

[72] Y. Dieudonné, F. Petit, and V. Villain. Leader election problem versus pattern formation problem. In *International Symposium on Distributed Computing (DISC)*, pages 267–281, 2010.

[73] E. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communication of the ACM (Commun. ACM)*, 17(11):643–644, 1974.

[74] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms (J. Algorithms)*, 51(1):38–63, 2004.

[75] S. Dobrev, N. Santoro, and W. Shi. Scattered black hole search in an oriented ring using tokens. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–8, 2007.

[76] S. Dolev. *Self-stabilization.* MIT Press, March 2000.

[77] S. Dubois and R. Guerraoui. Introducing speculation in self-stabilization: an application to mutual exclusion. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 290–298, 2013.

[78] S. Dubois, M.-H. Kaaouachi, and F. Petit. Enabling minimal dominating set in highly dynamic distributed systems. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 51–66, 2015.

[79] T. Erlebach, M. Hoffmann, and F. Kammer. On temporal graph exploration. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 444–455, 2015.

[80] A. Ferreira. On models and algorithms for dynamic communication networks: The case for evolving graphs. In *rencontres francophones sur les Aspects Algorithmiques des Telecommunications (ALGOTEL)*, 2002.

[81] M. Fischer, D. Jung, and F. Meyer auf der Heide. Gathering anonymous, oblivious robots on a grid. In *International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*, pages 168–181, 2017.

[82] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (J. ACM)*, 32(2):374–382, 1985.

[83] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 105–118, 2007.

[84] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. *Theoretical Computer Science (Theor. Comput. Sci.)*, 411(14-15):1583–1598, 2010.

[85] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. How many oblivious robots can explore a line. *Information Processing Letters (Inf. Process. Lett.)*, 111(20):1027–1031, 2011.

[86] P. Flocchini, M. Kellett, P. C. Mason, and N. Santoro. Fault-tolerant exploration of an unknown dangerous graph by scattered agents. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 299–313, 2012.

[87] P. Flocchini, M. Kellett, P. C. Mason, and N. Santoro. Finding good coffee in paris. In *International Conference on Fun with Algorithms (FUN)*, pages 154–165, 2012.

[88] P. Flocchini, M. Kellett, P. C. Mason, and N. Santoro. Searching for black holes in subways. *Theory of Computing Systems (Theory Comput. Syst.)*, 50(1):158–184, 2012.

[89] P. Flocchini, B. Mans, and N. Santoro. Exploration of periodically varying graphs. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 534–543, 2009.

[90] P. Flocchini, B. Mans, and N. Santoro. On the exploration of time-varying networks. *Theoretical Computer Science (Theor. Comput. Sci.)*, 469:53–68, 2013.

[91] P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta. Distributed computing by mobile robots: uniform circle formation. *Distributed Computing*, 30(6):413–457, 2017.

[92] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 93–102, 1999.

[93] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science (Theor. Comput. Sci.)*, 337(1-3):147–168, 2005.

[94] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science (Theor. Comput. Sci.)*, 407(1-3):412–447, 2008.

[95] P. Flocchini, N. Santoro, G. Viglietta, and M. Yamashita. Rendezvous of two robots with constant memory. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 189–200, 2013.

[96] K.-T. Foerster and R. Wattenhofer. Lower and upper competitive bounds for online directed graph exploration. *Theoretical Computer Science (Theor. Comput. Sci.)*, 655:15–29, 2016.

[97] P. Fraigniaud and D. Ilcinkas. Digraphs exploration with little memory. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 246–257, 2004.

[98] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science (Theor. Comput. Sci.)*, 345(2-3):331–344, 2005.

[99] T. Gotoh, Y. Sudo, F. Ooshita, H. Kakugawa, and T. Masuzawa. Group exploration of dynamic tori. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 775–785, 2018.

[100] S. Guilbault and A. Pelc. Gathering asynchronous oblivious agents with local vision in regular bipartite graphs. *Theoretical Computer Science (Theor. Comput. Sci.)*, 509:86–96, 2013.

[101] F. Harary and G. Gupta. Dynamic graph models. *Mathematical and Computer Modelling*, 25(7):79–87, 1997.

[102] A. Heriban, X. Défago, and S. Tixeuil. Optimally gathering two robots. In *International Conference on Distributed Computing and Networking (ICDCN)*, pages 3:1–3:10, 2018.

[103] S.-T. Huang and N.-S. Chen. A self-stabilizing algorithm for constructing breadth-first trees. *Information Processing Letters (Inf. Process. Lett.)*, 41(2):109–117, 1992.

[104] D. Ilcinkas, R. Klasing, and A. M. Wade. Exploration of constantly connected dynamic graphs based on cactuses. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 250–262, 2014.

[105] D. Ilcinkas and A. M. Wade. On the power of waiting when exploring public transportation systems. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 451–464, 2011.

[106] D. Ilcinkas and A. M. Wade. Exploration of the T-interval-connected dynamic graphs: The case of the ring. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 13–23, 2013.

[107] T. Izumi, Z. Bouzid, S. Tixeuil, and K. Wada. Brief announcement: The bg-simulation for byzantine mobile robots. In *International Symposium on Distributed Computing (DISC)*, pages 330–331, 2011.

[108] T. Izumi, D. Kaino, M. Gradinariu Potop-Butucaru, and S. Tixeuil. On time complexity for connectivity-preserving scattering of mobile robots. *Theoretical Computer Science (Theor. Comput. Sci.)*, 738:42–52, 2018.

[109] T. Izumi, S. Souissi, Y. Katayama, N. Inuzuka, X. Défago, K. Wada, and M. Yamashita. The gathering problem for two oblivious robots with unreliable compasses. *SIAM Journal on Computing (SIAM J. Comput.)*, 41(1):26–46, 2012.

[110] T. Izumi, Y. Yamauchi, and S. Kamei. Brief announcement: Mobile agent rendezvous on edge evolving rings. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 92–94, 2012.

[111] B. Katreniak. Convergence with limited visibility by asynchronous mobile robots. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 125–137, 2011.

[112] D. Kempe, J. M. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences (JCSS)*, 64(4):820–842, 2002.

[113] R. Klasing, E. Markou, and A. Pelc. Gathering asynchronous oblivious mobile robots in a ring. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 744–753, 2006.

[114] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. L. Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Transactions on Computer Systems (ACM Trans. Comput. Syst.)*, 27(4):7:1–7:39, 2009.

[115] D. R. Kowalski and A. Malinowski. How to meet in anonymous network. *Theoretical Computer Science (Theor. Comput. Sci.)*, 399(1-2):141–156, 2008.

[116] F. Kuhn, N. A. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *Symposium on the Theory of Computing (STOC)*, pages 513–522, 2010.

[117] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering (TSE)*, 3(2):125–143, 1977.

[118] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[119] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (ACM Trans. Program. Lang. Syst.)*, 4(3):382–401, 1982.

[120] M. Latapy, T. Viard, and C. Magnien. Stream graphs and link streams for the modeling of interactions over time. Technical report, arXiv 1710.04073, 2017.

[121] X. Li, M. F. Ercan, and Y.-F. Fung. Decentralized control for swarm flocking in 3d space. In *International Conference on Intelligent Robotics and Applications (ICIRA)*, pages 744–754, 2009.

[122] G. A. Di Luna, P. Flocchini, L. Pagli, G. Prencipe, N. Santoro, and G. Viglietta. Gathering in dynamic rings. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 339–355, 2017.

[123] G. A. Di Luna, P. Flocchini, N. Santoro, and G. Viglietta. Turingmobile: A turing machine of oblivious mobile robots with limited visibility and its applications. In *International Symposium on Distributed Computing (DISC)*, page to appear, 2018.

[124] G. Antonio Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and M. Yamashita. Meeting in a polygon by anonymous oblivious robots. In *International Symposium on Distributed Computing (DISC)*, pages 14:1–14:15, 2017.

[125] G. Di Luna, S. Dobrev, P. Flocchini, and N. Santoro. Live exploration of dynamic rings. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 570–579, 2016.

[126] O. Michail and P. G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science (Theor. Comput. Sci.)*, 634:1–23, 2016.

[127] R. O'Dell and R. Wattenhofer. Information dissemination in highly dynamic graphs. In *DIALM-POMC Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 104–110, 2005.

[128] F. Ooshita and A. K. Datta. Feasibility of weak gathering in connected-over-time dynamic rings. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, page to appear, 2018.

[129] F. Ooshita and S. Tixeuil. On the self-stabilization of mobile oblivious robots in uniform rings. *Theoretical Computer Science (Theor. Comput. Sci.)*, 568:84–96, 2015.

[130] D. Peleg. Distributed coordination algorithms for mobile robot swarms: New directions and challenges. In *International Workshop on Distributed Computing (IWDC)*, pages 1–12, 2005.

[131] R. De Prisco, B. W. Lampson, and N. A. Lynch. Revisiting the PAXOS algorithm. *Theoretical Computer Science (Theor. Comput. Sci.)*, 243(1-2):35–91, 2000.

[132] C. Shannon. Presentation of a maze-solving machine. *Conference of the Josiah Macy, Jr. Foundation*, pages 173–180, 1951.

[133] M. Shibata, T. Mega, F. Ooshita, H. Kakugawa, and T. Masuzawa. Uniform deployment of mobile agents in asynchronous rings. *Journal of Parallel and Distributed Computing (J. Parallel Distrib. Comput.)*, 119:92–106, 2018.

[134] S. Souissi, X. Défago, and M. Yamashita. Using eventually consistent compasses to gather memory-less mobile robots with limited visibility. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(1):9:1–9:27, 2009.

[135] G. Di Stefano and A. Navarra. Optimal gathering of oblivious robots in anonymous graphs and its application on trees and rings. *Distributed Computing*, 30(2):75–86, 2017.

[136] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots. In *International Colloquium on Structural Information & Communication Complexity (SIROCCO)*, pages 313–330, 1996.

[137] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing (SIAM J. Comput.)*, 28(4):1347–1363, 1999.

[138] S. Tixeuil. *Algorithms and Theory of Computation Handbook*, chapter Self-stabilizing Algorithms, pages 26.1–26.45. Chapman & Hall. CRC Press, Taylor & Francis Group, November 2009.

[139] T. Uehara, Y. Yamauchi, S. Kijima, and M. Yamashita. Plane formation by semi-synchronous robots in the three dimensional euclidean space. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 383–398, 2016.

[140] N. H. Vaidya and D. K. Pradhan. Degradable agreement in the presence of byzantine faults. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 237–244, 1993.

[141] N. Xiong, J. He, Y. Yang, Y. He, T.-H. Kim, and C. Lin. A survey on decentralized flocking schemes for a set of autonomous mobile robots (invited paper). *Journal of Communications (JCM)*, 5(1):31–38, 2010.

[142] Y. Yamauchi, T. Izumi, and S. Kamei. Mobile agent rendezvous on a probabilistic edge evolving ring. In *International Conference on Networking and Computing (ICNC)*, pages 103–112, 2012.

[143] Y. Yamauchi, T. Uehara, S. Kijima, and M. Yamashita. Plane formation by synchronous mobile robots in the three-dimensional euclidean space. *Journal of the ACM (J. ACM)*, 64(3):16:1–16:43, 2017.